UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

ADAPTACIÓN DEL ALGORITMO DE ABEJAS (ABC) PARA LA IDENTIFICACIÓN DE GRUPOS DE USUARIOS EN REDES EGO

David Vicente López
Tutor: Antonio González Pardo
Ponente: David Camacho Fernández

Julio 2019

ADAPTACIÓN DEL ALGORITMO DE ABEJAS (ABC) PARA LA IDENTIFICACIÓN DE GRUPOS DE USUARIOS EN REDES EGO

AUTOR: David Vicente López
TUTOR: Antonio González Pardo

Applied Intelligence and Data Analysis Group (AIDA)
Dpto. Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio de 2019

Resumen (castellano)

Actualmente, el uso de las redes sociales abarca uno de los principales cambios que está experimentando la sociedad de cara a la manera que tenemos los humanos de relacionarnos entre nosotros. Este cambio deja un importante rastro de información que se puede usar para diversos fines, uno de ellos es la detección de comunidades.

Uno de los problemas recurrentes a la hora de tratar esta información es lo costoso que puede ser computacionalmente un algoritmo convencional. Por esta razón se están ideando algoritmos más novedosos como los algoritmos heurísticos. En este documento se desarrollará este problema de manera más extensa.

El Trabajo de Fin de Grado llevado a cabo se enmarca en el ámbito de la algoritmia en el tratamiento de grafos, más concretamente orientado al problema de detección de comunidades. Se ha diseñado e implementado un software capaz de reproducir y medir distintos parámetros con el fin de poner a prueba un algoritmo bio-inspirado llamado ***Atificial Bee Colony***.

Este algoritmo de enjambre permite encontrar una solución eficaz al problema que se le plantea utilizando a unos agentes llamados abejas, que a su vez podrán variar su comportamiento basado en el tipo de abeja que sean. Siendo posible ser abeja obrera (*employee*), observadora (*onlooker*) o exploradora (*scout*).

Se ha obtenido y asimilado información de varios documentos de diferentes autores, donde se exponen ciertos métodos y algoritmos para resolver los problemas que encontramos a lo largo del trabajo realizado. Posteriormente se ha implementado un software para probar el funcionamiento de este algoritmo basado en varios principios de la documentación. Se han llevado a cabo pruebas y experimentos para obtener diversos resultados y, de esta manera, poder encontrar los parámetros y características que más benefician al funcionamiento del algoritmo.

Finalmente se expondrán las conclusiones sobre el uso del algoritmo en el ámbito de la detección de comunidades. También se tratarán sus puntos fuertes y no tan fuertes y se abrirán nuevas áreas de trabajo respecto al proyecto realizado.

Palabras clave (castellano)

Detección de comunidades, ABC, Algoritmo de enjambre, Grafo, Algoritmo heurístico, Artificial Bee Colony, Redes sociales, Agrupamiento, Abejas.

Abstract (English)

Currently, the use of social networks covers one of the main changes that society is experiencing in the way that humans have to relate to each other. This change leaves an important trail of information that can be used for various purposes, one of which is the detection of communities.

One of the recurring problems when dealing with this information is how expensive a conventional algorithm can be computationally. For this reason they are devising newer algorithms such as heuristic algorithms. In this document, this problem will be developed more extensively.
The Final Degree Project carried out is part of the field of algorithm in the treatment of graphs, more specifically oriented to the problem of community detection. A software capable of reproducing and measuring different parameters has been designed and implemented in order to test a bio-inspired algorithm called ***Atificial Bee Colony***.

This swarm algorithm allows finding an effective solution to the problem that is posed using agents called bees, which in turn can vary their behavior based on the type of bee they are. Being possible to be employee, onlooker or scout.

Information has been obtained and assimilated from several documents of different authors, where certain methods and algorithms are exposed to solve the problems that we find throughout the work carried out. Subsequently, software has been implemented to test the operation of this algorithm based on several principles of documentation. Tests and experiments have been carried out to obtain different results and, in this way, to be able to find the parameters and characteristics that most benefit the operation of the algorithm.
Finally, the conclusions on the use of the algorithm in the field of community detection will be presented. Their strengths and not so strong will also be discussed and new areas of work will be opened regarding the project carried out.

Keywords (inglés)

Detection of communities, ABC, Swarm Algorithm, Graph, Heuristic Algorithm, Artificial Bee Colony, Social Networks, Grouping, Bees.

Agradecimientos

En primer lugar me gustaría agradecer mi trabajo a la persona que mayor mérito tiene de todas las personas mencionadas en este apartado, quien no podría haber sido un mejor tutor, Álvaro González Pardo, a quien debo un gran número de horas y esfuerzo invertido en investigar, alentar y facilitarme las cosas cuando el camino se ponía duro. Gracias por haberme brindado la oportunidad de poder colaborar en este interesantísimo proyecto con unos fines tan nobles como los que tiene. Ha sido un placer haber trabajado contigo.

También quiero agradecer a mi padre Francisco Vicente y mi madre Yolanda López por todo el apoyo que me han brindado no solo en este proyecto, si no a lo largo de mi vida hasta llegar al punto en el que me encuentro ahora mismo. Son sin duda dos personas excepcionales que merecen una gran parte de todo el éxito que consiga.
Por último pero no menos importante querría agradecer a mis compañeros y compañeras de clase con quien he tenido la suerte de compartir mis inquietudes y varios de los mejores momentos hasta la fecha. Ha sido un verdadero placer trabajar en equipo y con gente tan valiosa e inteligente.

Más allá de los conocimientos adquiridos en la carrera, me llevo buenos amigos. Para este último apartado me gustaría mencionar a Alberto Prudencio, Junior Inván Soler, Eamon Thornton, Adrián Pertejo, Marcos Redondo, Adrián Navas y Rubén Sastre por todo el apoyo, ayuda y buenos momentos que he compartido con ellos y por encima de eso las cosas que me han enseñado por las cuales ahora soy una persona distinta y mejorada a la que entró en esta facultad.

Muchas gracias a todos y a todas por estos años.
David Vicente López
Junio, 2019

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

1.1 Motivación

En los últimos años la tecnología y más concretamente el acceso a internet nos ha permitido modificar la manera en la que nos relacionamos: este es un hecho con un grandísimo abanico de temas a tratar, pero para esta puesta en escena nos centraremos en un ámbito concreto: Las redes sociales.

Tras realizar una búsqueda sobre el tema [1], de 2014 hasta el año actual como se puede observar en la siguiente imagen ha crecido el uso de estas herramientas en un 13,5% de media respecto al año anterior. Y si asimilamos los últimos 5 años casi se duplica el número de usuarios de las redes sociales.

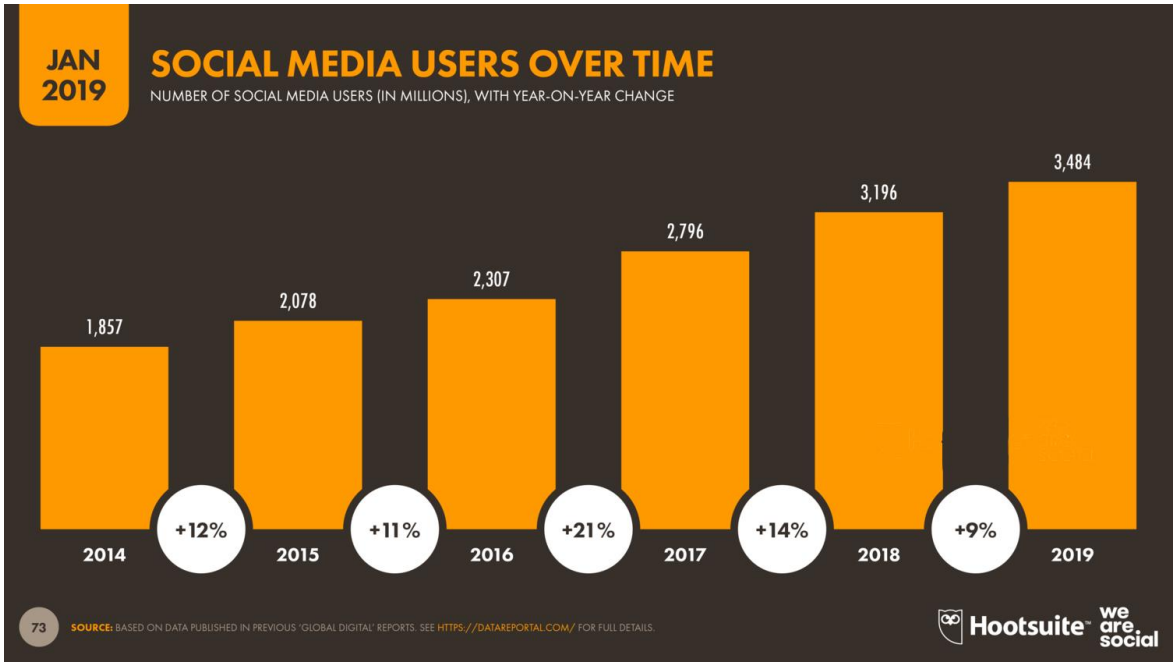

Figura 1-1: Crecimiento de los últimos 5 años en redes sociales

Este hecho ha cambiado nuestra manera de comunicarnos y ha traído consigo nuevos métodos de obtención de información, publicidad, detección de targets para marketing, encuestas… Junto a muchos otros métodos que no están desarrollados o faltan por descubrir. Este punto de desarrollo e innovación ha despertado el interés de numerosas entidades (empresas, grupos de investigación, seguridad, gobierno…) por el desarrollo de esta tecnología capaz de cuantificar, medir y extraer la información tan abundante que nos proporcionan las redes sociales.
Es un hecho que cuantos más usuarios tengamos en las redes sociales, más se incrementará el número de interacciones y de relaciones entre los usuarios, y por lo tanto esta información será cada vez es más grande y más difícil de medir y gestionar, generando una serie de problemas a los que la algoritmia en el ámbito de las TIC puede otorgar solución.

Las posibilidades del software para obtener información de redes sociales son muy abundantes ya que por ofrecen posibilidades como:

- Estudiar varias características de la red y desarrollar teorías sobre fenómenos en la misma.
- Identificar nodos o comunidades clave (personas o comunidades de gran influencia).
- Monitorización de la evolución de la red y de cómo fluye la información por la misma.
- Detectar comunidades de individuos.

En este Trabajo Fin de Grado (TFG), nos centraremos en las tareas de detección de comunidades que permite agrupar los usuarios con mayor número de similitudes entre sí en la misma comunidad mientras que los usuarios más diferenciados estarán en comunidades diferentes.
Para la detección de dichas comunidades estudiaremos la viabilidad de aplicar un algoritmo basado en el funcionamiento de las colmenas de abejas llamado *Artificial Bee Colony* (*ABC*). Cabe destacar que dicho algoritmo se ha desarrollado inicialmente para la resolución de problemas de optimización, por lo que en este TFG ha sido necesaria su adaptación para poder realizar tareas de detección de comunidades.

1.2 Objetivos

El trabajo presentado tiene marcados los objetivos:

El trabajo presentado tiene marcados los siguientes objetivos:

1. Comprender el funcionamiento de diferentes *papers*, o trabajos de investigación, sobre el análisis de redes de grafos con la finalidad de crear una representación de una red social y desarrollar un algoritmo usando esa representación.

2. Generar una representación de datos de una red social a partir de unos datos proporcionados, así como el software necesario para leer, transformar y gestionar esos datos, una parte necesaria para probar el funcionamiento del algoritmo.

3. Crear a partir de las explicaciones y el pseudocódigo expuesto en uno de los documentos un algoritmo inspirado en el funcionamiento de las colonias de abejas que nos permitirá de manera heurística detectar las comunidades.

4. Analizar y realizar experimentos para evaluar el rendimiento y la efectividad del algoritmo desarrollado, y estudiar la viabilidad del algoritmo para resolver el problema de la detección de comunidades.

1.3 Metodología y Planificación del trabajo

Para alcanzar todos los objetivos del trabajo se ha seguido una metodología mayormente secuencial, pudiendo paralelizar ciertas tareas de redacción de la memoria. Para describir mejor la realización del trabajo se ha creado el siguiente diagrama de Gantt:

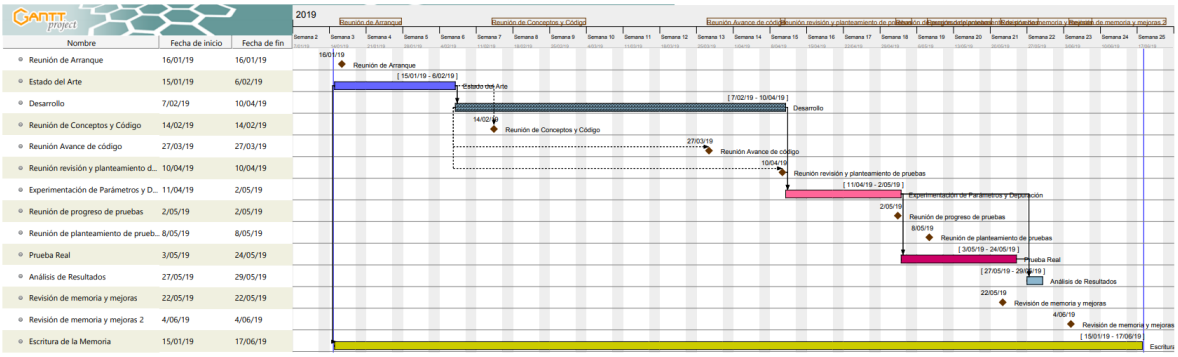

Figura 1-2: Diagrama de Gantt

- **Estado del arte**: El primer paso para comenzar el trabajo sería documentarse sobre la situación actual de la detección de comunidades en redes sociales. Para ello se leen artículos como los posteriormente descritos no sólo para conocer su situación sino para obtener nuevos conocimientos que posteriormente serán empleados en la fase de implementación.

- **Desarrollo del Software**: La creación de un software que permite leer información de una base de datos de una red social, generar un grafo con dicha información y por último adaptarlo la solución propuesta. Esta adaptación consiste en adaptar ABC con el problema de la detección de comunidades. El apartado sería lo que se define en este documento como la adaptación del algoritmo ABC al problema de la adaptación de comunidades (Sección 3)

- **Definición del valor de los parámetros**: los algoritmos heurísticos se caracterizan, en parte, por la gran cantidad de parámetros que afectan al rendimiento de dicho algoritmo. Por ello, es necesario realizar un estudio para fijar minuciosamente el valor de cada parámetro de manera que el rendimiento del algoritmo sea lo más óptimo posible.

- **Prueba Real**: Tras los pasos anteriores y con un software funcionando y configurado de la manera más eficiente entre las estudiadas en el paso anterior, se procede a ejecutar el algoritmo sobre el dataset en cuestión para extraer las comunidades que componen dichas redes.

- **Análisis de Resultados**: Una vez tenemos los resultados de los experimentos y las distintas configuraciones es hora de compararlos, se realiza un estudio del comportamiento del algoritmo exponiendo los datos de una manera clara e ilustrativa.

- **Escritura de la memoria**: Tras los pasos realizados con anterioridad, se redacta la memoria de Trabajo de Fin de Grado, en la que se pone en comparación el previo estado de arte con los avances realizados a lo largo de este trabajo.

1.4 Organización de la memoria

La memoria consta de los siguientes capítulos:

- **Sección 1: Introducción**. Aquí describimos los conceptos clave y establecemos unos principios y aspectos generales para la motivación y organización del trabajo realizado

- **Sección 2: Estado del Arte**. En esta sección describimos las herramientas y los conceptos que se han usado y el estado actual de las mismas.

- **Sección 3: Adaptación del algoritmo ABC al problema de la adaptación de comunidades**. En esta sección se describe el diseño del algoritmo así como su su implementación y adaptación al problema que se desea resolver.

- **Sección 4: Fase de Experimentación**. En este bloque se describe el dataset seleccionado, así como la manera en la que se han decidido los parámetros y las pruebas que se han realizado.

- **Sección 5: Conclusiones y trabajo futuro**. En esta sección se desarrollan las ideas que se han evolucionado durante la elaboración del trabajo. También se exponen posibles puntos de partida para la mejora o el desarrollo del trabajo en un futuro.

2 Estado del arte

En el siguiente apartado se describirán diferentes temáticas y aspectos importantes en lo que nos tendremos que basar para la posterior realización de nuestra solución al problema de detección de comunidades.

2.1 Redes sociales y su representación

Para realizar este trabajo es necesario asimilar el concepto abstracto de una red social así como su importancia en el mundo actual y las potenciales aplicaciones que se pueden realizar para extraer información valiosa de la red. Por otro lado deberemos también saber la representación con la que trabajaremos a lo largo del TFG.

2.1.1 Redes sociales

En la actualidad, y como ya hemos hablado en el apartado de motivación, las redes sociales se podrían definir casi como figuras abstractas que representan en diferentes ámbitos las relaciones entre las personas.
Llevado al ámbito filosófico podríamos hablar de las relaciones humanas como un paradigma que mueve el mundo y la sociedad humana, algo tan sumamente valioso como la comunicación que lleva dirigiendo al mundo desde hace siglos, las redes sociales son una expresión de esas relaciones que podemos cuantificar, medir y evaluar de manera "sencilla" al tener los datos recopilados y ordenados.

Si hablamos de intereses y teniendo en cuenta los datos expuestos en el apartado 1, cuantos más usuarios existen, el número de interacciones entre ellos aumenta también de manera casi exponencial. Lo que provoca un aumento de datos espectacular, que no para de crecer a medida que el tiempo transcurre y que van aumentando sus usuarios.

Toda esta información es, como ya hemos hablado anteriormente, en muchos aspectos interesante a la hora de conocer las relaciones, estructuras sociales, comunidades y muchos otros ámbitos. Diferentes entidades como empresas, equipos de investigación, gobiernos, organizaciones… están sumamente interesada en no sólo acceder a estos datos, sino también en tener la capacidad de tratarlos correctamente de distintas maneras para obtener todo su potencial en forma de estrategias de actuación, estadísticas, sistemas de predicción, detección de puntos de interés… Y una infinidad más de herramientas que podrían ser desarrolladas a partir de este ámbito.

Por otro lado, a mayor número de interacciones y mayor número de usuarios, el valor de la información almacenada en una red social se ve disparada pero a su vez surgen problemas al tener que gestionar y manejar cada vez más ingentes cantidades de datos e información, mayor número de problemas e incluso mayores "capas" de información debido a la creciente complejidad de las redes sociales.

2.1.2 Representación de redes sociales

Debemos tener en cuenta que una red social es capaz de expresarse mediante un grafo **G(V, E)** donde sus nodos o vértices (**V**) son los usuarios de esa red social y las aristas (**E**) son las relaciones entre esos usuarios. Las características de este grafo nos hablarán de la estructura y las peculiaridades de la red social como la densidad, grado de conexión, ramificaciones, agrupaciones…

Debemos tener en cuenta que según la red social este grafo podría variar algunas de sus características, por ejemplo, su **dirección**. Dependiendo del tipo de red social con la que estemos tratando presentan una única relación entre 2 usuarios, por lo que el grafo de esa red social sería un **grafo no dirigido** y sólo requeriría una arista para describir esa relación. Sin embargo existen otro tipo de redes sociales donde esta relación se expresa como un interés de un usuario por otro, y no tiene que ser bidireccional (o recíproco), por lo que surge una dirección en el grafo que expresa un interés no mutuo. En estos casos se utiliza un modelo distinto de grafo, el **grafo dirigido**. Este ejemplo es uno de muchos por los que dependiendo de la red social en la que nos encontremos pueden variar distintas características de la misma.

Como antes hemos mencionado, las redes sociales cada vez van aumentando en complejidad debido a que ya no solamente existen las relaciones de "contacto": véase *usuario1* **es amigo o sigue** a *usuario2*. En la actualidad existen una gran variedad de interacciones distintas entre usuarios. Llevado al ámbito práctico un **"like" de un usuario a la publicación de otro usuario** es un tipo distinto de interacción que una mención del *usuario2* escrita por el *usuario1* en un comentario en un post de un tercer usuario.

Como se puede ver a partir de este ejemplo práctico, la complejidad de las redes sociales no se queda en "una capa" si no que existen distintos niveles de extracción de información. Sin embargo, todos ellos están relacionados entre sí.

Un ejemplo ilustrativo sería la siguiente imagen:

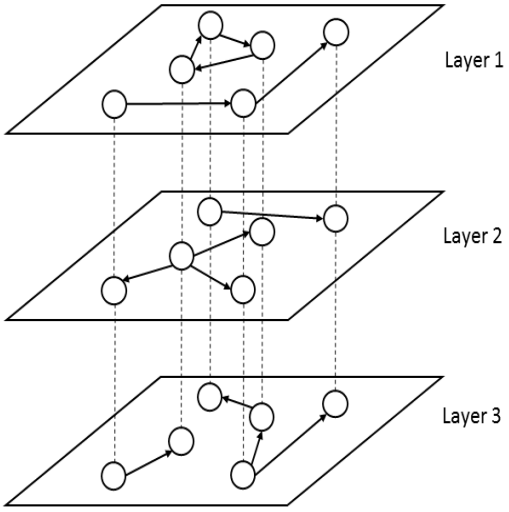

Figura 2-1: Percepción multicapa de Red Social

En la figura anterior, si se plantean por ejemplo:

- **Layer 1:** "Me gusta" de una red social
- **Layer 2:** Comentario de un usuario a la publicación de otro usuario.
- **Layer 3:** Mención de un usuario a otro en una publicación de un tercero

Como podemos observar estas relaciones entre los usuarios aportan informaciones muy distintas aunque sumamente relacionadas entre sí. Con este ejemplo probamos que no solamente las relaciones entre usuarios pueden ser de "amistad" o "se siguen" existe una gran variedad de información una vez hemos pasado esa barrera superficial.

2.1.3 Aplicaciones sobre redes sociales

Cuando hablamos de tratar estas redes sociales tratamos un campo bastante amplio debido a que las redes sociales ofrecen un gran abanico de estudios que nos permiten extraer distintos tipos de información y con muy distintas finalidades.

Sin ir más lejos hemos hablado en secciones anteriores de las propiedades de la red o de los tipos de representación que se le pueden asignar, pero la realidad es que de por sí las propiedades de la red nos pueden aportar una grandísima información sobre la evolución de esa red social y cómo está estructurada si la extrapolamos al mundo real. Por otro lado, la representación de tipo grafo es únicamente una elección para llegar al objetivo en este trabajo, pero existen otros modos distintos y en ocasiones más complejos de representación y que de por sí aportan información distinta.

Otra posibilidad es el estudio de usuarios o de temas que tienen gran influencia en esa red, los llamados "*topics*". Y también se puede analizar la evolución de estos *topics* a lo largo del tiempo. Esta posibilidad podría ser útil para estudiar la propagación de noticias, para realizar campañas de marketing o sencillamente para conocer cómo un determinado tema influye sobre una comunidad para su posible clasificación.

También se puede estudiar la identificación de personajes clave en la red, ya que no es lo mismo tener una posición estratégica en esa red que tener influencia, o que tener un gran número de contactos.

Por último, otra de las muchas posibilidades que ofrece el estudio de las redes sociales es la detección de comunidades, basado en calcular la cómo de buena es una solución donde que un usuario pertenezca a una comunidad con características afines. Esta estimación de cómo de buena es una solución se desarrollará más en profundidad a continuación, pero permitiría tener parámetros para clasificar a los usuarios de una misma comunidad, esto es interesante para realizar perfilamientos, marketing y operaciones de seguridad entre otras muchas cosas.

2.1.4 Redes Ego

Al manejar un tipo tan complejo de información como un grafo representativo de una red social, podemos obtener unas cantidades muy grandes de datos.
Al no saber dónde cortar esa muestra ni con qué criterio establecer los nodos y las conexiones que van a entrar en nuestro conjunto de datos, debemos tener una herramienta de corte para el desarrollo que va a venir implícito en la realización del trabajo. Por lo tanto se ha establecido como criterio de aislamiento las Ego-Networks o Redes Ego.

Estas redes se basan en escoger un número determinado de "Egos" que son usuarios escogidos mediante un criterio de interés específico como centros de la muestra, es decir, que el experimento y la muestra giran en todo a esos usuarios. Esto tiene una explicación, cada ego posee un número determinado de vecinos a los que llamaremos "Alters" y estos nodos junto a los Egos serán escogidos para realizar la prueba.

De esta manera se añaden unas características a la red, y es que todos los alters independientemente de su relación entre ellos tienen una relación con el ego, debido a que como muestra estamos cogiendo los vecinos del mismo, por lo tanto a la hora de procesar la muestra debemos ignorar estas relaciones entre el ego y los alters para obtener unos resultados sin distorsionar por este hecho.

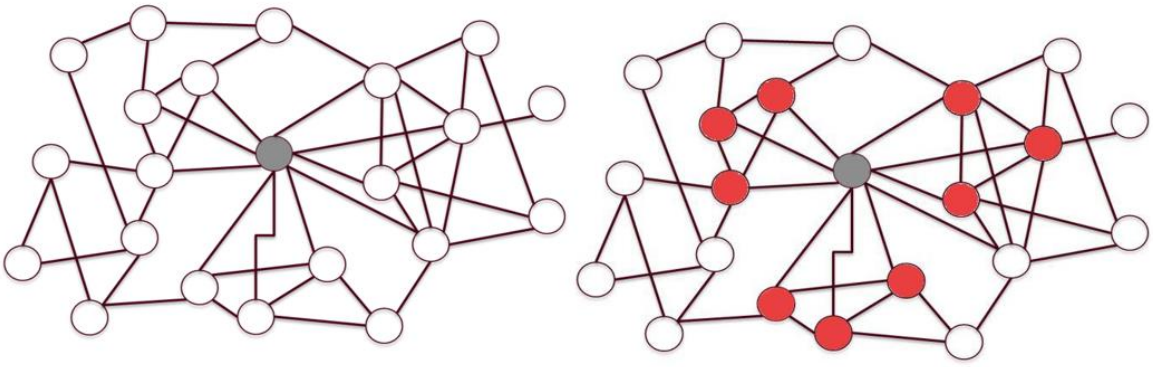

Figura 2-2: Selección de Alters

En la anterior figura de una red social de prueba, podemos observar que estableciendo un ego (punto gris) y obteniendo únicamente sus vecinos (alters, en rojo) tenemos un grafo bastante simplificado y más fácilmente tratable, quedando un grafo como se expone a continuación:

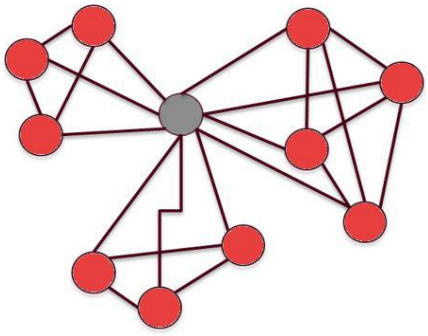

Figura 2-3: Red Ego

2.1.5 Comunidades

Una comunidad es un grupo de nodos correlacionados altamente entre sí y con una conexión más débil con el resto de comunidades. Que estas a su vez comparten una serie de características por las cuales ese conjunto de nodos se "conocen" por una serie de circunstancias comunes.

Estas comunidades se tratan de **subgrafos dentro del propio grafo** que es la red social. La idea es que a simple vista se podría ver una comunidad bien definida diferenciándose de otra ya que la densidad entre las relaciones internas de la comunidad es muy densa, mientras que las relaciones externas de esta red son de menor densidad. Si expusiésemos esta comunidad como un grafo pintado diferenciaríamos **distintos grupos o clusters**.
Extrapolado al lenguaje de las redes sociales, una comunidad tiene un punto de contacto donde se han conocido, por ejemplo: haber compartido clase, trabajado juntos, compartir un hobby, vivir cerca unos de otros, estar en el mismo grupo de investigación, asistir a determinado evento… En definitiva, una circunstancia común entre varios usuarios.

Para distinguir una comunidad de otra en un grafo de red social debemos tener un método que nos proporcione información sobre su densidad interna (el número de aristas que existen entre nodos de esta comunidad) y su densidad externa (el número de nodos que salen de esa comunidad para comunicarse con miembros externos a esa comunidad) pudiendo sacar así un dato limpio que nos exprese la probabilidad de que un grupo de nodos se trate de una comunidad bien definida.

La siguiente fórmula nos permite obtener una puntuación o "score" para medir estas características, cuanto más alto sea dicho score más probable será de tratarse de una comunidad.

$$D_{int} = \frac{Número\ de\ aristas\ internas}{\frac{Aristas\ de\ la\ comunidad - 1}{2}}$$

$$D_{ext} = \frac{Número\ de\ aristas\ externas}{Número\ de\ comunidades\ total - Aristas\ de\ la\ comunidad}$$

A la hora de detectar una comunidad, nuestra intención debe ser maximizar la densidad interna (alta cohesión entre los miembros de la comunidad) y minimizar la densidad externa (baja conexión con el exterior) por lo tanto nuestro Score o puntuación a la hora de decidir si se trata de una comunidad debe ser:

$$Score = D_{int} - D_{ext}$$

Cuando realicemos un **sumatorio de este score** para todas las comunidades que pretendemos clasificar nos otorgará una **puntuación de la solución al problema** que nos permitirá medir puntos como el avance del score en secuencias, variaciones, sustituciones y principalmente cómo de buena es nuestra solución.

2.2 Algoritmo para la detección de comunidades.

2.2.1 Decisión Heurística

En nuestro caso al tener como punto de partida una red social (expresada según se ha explicado en el anterior apartado) y empezar desde 0 la clasificación se trata de un problema de detección y clasificación de grupos o clusters con indicadores que midan esa cohesión interna y externa explicada anteriormente, **en definitiva, el objetivo principal es mejorar el score que hemos definido para aumentar las probabilidades de que sea una solución verídica**.

Este score como ya hemos visto viene dado por dos componentes, una que queremos disminuir y otra que queremos aumentar para la mejor definición de una comunidad, por lo tanto podría decirse que tratamos con un pariente cercano a la decisión Min-Max.

Por estas razones la complejidad con la que debemos lidiar nuestro problema se eleva hasta NP-Hard (No determinista), entonces, se decide seguir un **enfoque heurístico y evolutivo**. Con este nuevo enfoque promovemos que una solución inicial se va auto-perfeccionando basándose en los principios de la Inteligencia Artificial y eligiendo siempre una solución mejor según pasan sus generaciones.

Este tipo de algoritmos suelen obtener una serie de "participantes" representando posibles soluciones de un problema que compiten entre ellas para generar a su vez nuevas y mejoradas soluciones y descartando las menos efectivas, creando así generaciones que, siguiendo el método evolutivo, superan a sus predecesoras.

Como conclusión este tipo de algoritmos no ofrecen una sola solución como resultado de un input, si no que mediante ensayos y comparaciones con otras soluciones ya proporcionadas o con sus versiones anteriores se establece la valía de esa solución.

2.2.2 Tipos de Algoritmos Heurísticos Posibles

En nuestro caso al tener como punto de partida una red social (expresada según se ha explicado en el anterior apartado) y empezar desde 0 la clasificación, se trata de un problema de detección y clasificación de grupos o *clusters* con indicadores que midan esa cohesión interna y externa explicada anteriormente.
En definitiva, el objetivo principal es mejorar el score que hemos definido para aumentar las probabilidades de que sea una solución verídica.

Este score como ya hemos visto viene dado por dos componentes, una que queremos disminuir y otra que queremos aumentar para la mejor definición de una comunidad, por lo tanto podría decirse que tratamos con un pariente cercano a la decisión Min-Max.

Por estas razones la complejidad con la que debemos lidiar nuestro problema se eleva hasta NP-Hard (No determinista), entonces, se decide seguir un **enfoque heurístico**. Con este nuevo enfoque promovemos que una solución inicial se va auto-perfeccionando basándose

en los principios de la Inteligencia Artificial y eligiendo siempre una solución mejor según pasan sus generaciones.

Este tipo de algoritmos suelen obtener una serie de "participantes" representando posibles soluciones de un problema que compiten entre ellas para generar a su vez nuevas y mejoradas soluciones y descartando las menos efectivas, creando así generaciones que, siguiendo el método evolutivo, superan a sus predecesoras.

Como conclusión este tipo de algoritmos no ofrecen una sola solución como resultado de un input, si no que mediante ensayos y comparaciones con otras soluciones ya proporcionadas o con sus versiones anteriores se establece la valía de esa solución.

A la hora de resolver este problema nos encontramos con varias vertientes de algoritmos heurísticos, también hemos mencionado que debe ser evolutivo, no que genere soluciones aleatorias para resolver el problema por "fuerza bruta".

La Inteligencia Computacional es una rama de este tipo de algoritmos que nos proporciona mecanismos de adaptación al problema, que en este caso se ve muy indicado para resolver el problema planteado. Dentro de esta rama podemos encontrar distintas sub-ramas válidas a la hora de intentar obtener resultados satisfactorios.

Entre otras técnicas como **redes neuronales** o **sistemas difusos** podemos encontrar los **algoritmos evolutivos**, centrados especialmente en el progreso de las soluciones planteadas. Dentro de este apartado podríamos hablar de los Algoritmos genéticos.

También podríamos hablar de tratar con **Inteligencia de Enjambre**, este campo en concreto será en el que más se profundice en este trabajo ya que podemos clasificar nuestro algoritmo con este concepto. La inteligencia de Enjambre nos permite estudiar el comportamiento colectivo en un sistema que no está centralizado. Estos algoritmos poseen un conjunto de agentes que actúan de manera sencilla que, conglomerándose forman una solución compleja a un problema.

Este tipo de inteligencia está típicamente enlazado con la naturaleza y los algoritmos **bioinspirados** de los cuales hablaremos a continuación, debido a que este tipo de patrones se dan en un sinfín de comportamientos en la naturaleza (crecimiento bacteriano, alineamiento de las aves en vuelo, colonias de hormigas…)

2.2.3 Algoritmos Bioinspirados

Los **algoritmos bioinspirados** son algoritmos basados en algún aspecto de cómo la naturaleza hace frente a un problema. Aunque esto pueda sonar tremendamente amplio, este tipo de algoritmos pueden ser desarrollados partiendo de muchas fuentes distintas, por ejemplo un comportamiento de un determinado animal (ya sea individual o colectivo) un patrón climático, un ciclo evolutivo…

Estos algoritmos obtienen las características y ciertos patrones de las herramientas de la naturaleza normalmente para resolver algún problema de optimización [2].

Estos algoritmos están bien valorados en este tipo de problemas por ventajas como:

- Extrapolación cómoda ya que son bastante genéricos y fácilmente modulables a distintos problemas con una adaptación poco costosa.
- Normalmente otorgan carias soluciones en lugar de una sola, lo que se llaman "poblaciones"
- Son normalmente paralelos, ya que trabajan con varias soluciones, esto permite dividir el trabajo y recorrer sistemas de decisión con mucha expansión a la hora de tomar decisiones.

No obstante debemos tener en cuenta sus puntos flacos ya que normalmente dependen de una **gran cantidad y variedad de parámetros** a los que reaccionan de maneras poco previsibles, la mayoría tienen **componentes aleatorios** que no **nos permiten llegar a una misma solución con una entrada igual** y nos encontramos con riesgo de la llamada **"convergencia prematura"** que limitaría una población convergiendo en un máximo local no encontrando de esa manera la **solución óptima.**

2.3 Algoritmo Artificial Bee Colony (ABC)

Las abejas tienen uno de los comportamientos sociales más complejos del mundo animal ya que su colonia se basa en clases de abejas. Este comportamiento está altamente organizado y estructurado de una manera no aleatoria: cada clase tiene una función y toda su jerarquía está orientada por los roles que se realizan acorde al trabajo, cada abeja cumple una función y se comunican entre ellas para ponerse de acuerdo con sus roles

Entre otras muchas posibles comunicaciones (como por ejemplo si se necesita más cera para construir el panal o si se necesita elevar la temperatura de la zona cercana a la reina) podemos encontrar la información que nos interesa para el desarrollo de nuestro algoritmo: **Dónde se encuentras las fuentes de alimento**. Esta información se muestra principalmente mediante bailes, ya que las feromonas están reservadas para comunicar las necesidades internas.

Una vez dentro del tipo de abeja obrera (Ni los zánganos ni la reina) encontramos con varios tipos de roles para las mismas. De manera natural varían a lo largo de su vida debido a que los 20 primeros días (de media) las abejas están dentro de la colmena produciendo cera, limpiando el panal y cuidando de las larvas. Una vez pasan estos días la mayoría de abejas salen al exterior de la colmena, pero diferenciando las tareas existen distintos tipos de roles entre las abejas obreras que veremos a continuación.

En este trabajo de fin de grado se trabajará con el algoritmo "***Artificial Bee Colony***" (***ABC***).
Nuestro **ABC** se trata de un algoritmo bioinspirado de enjambre que se basa en el comportamiento de las abejas de la miel a la hora de obtener fuentes de alimento. Este algoritmo fue propuesto por **primera vez en 2005 por Dervis Karaboga** [3] y nos permite manejar una serie de soluciones que irán cambiando con el tiempo siguiendo los cánones y el comportamiento de distintos tipos de abejas en una colmena.

Este algoritmo se podría definir como un algoritmo de enjambre con un grado de complejidad un poco más alto, debido a que el algoritmo como tal posee agentes de distinto "rol social" debido a que cada tipo de agente tiene una tarea distinta a otro.

En ABC trabajaremos con distintos tipos de agentes, o de ahora en adelante **abejas**, que dependiendo de su rol seguirán un comportamiento u otro. Estos agentes tratarán de conseguir la mejor fuente de comida que estará planteada como la mejor solución para el problema que queremos resolver.

De ahora en adelante nos referiremos a puntos de comida como posibles soluciones al problema que se ha planteado para que el algoritmo lo resuelva.
Este algoritmo se puede dividir en 3 fases secuenciales correspondientes a cada tipo de abeja siguiendo las clases que hemos mencionado anteriormente:

- **Abejas obreras:** Son asignadas a una solución para recoger el polen o alimento (traducido como obtener nuestro score mencionado anteriormente) y exploran la zona circundante a ese punto de comida. En caso de encontrar un punto de comida mejor (con una puntuación mejor al punto actual) cambian de posición e informan al resto de abejas.

- **Abejas observadoras:** Rastrean las inmediaciones de los puntos de alimento que se conocen gracias a las abejas obreras, en caso de encontrar un punto mejor, informan a la colmena del nuevo hallazgo, cambiando así la dirección de las abejas obreras.

- **Abejas exploradoras o scouts:** Estas abejas se dedicarán a recorrer un punto aleatorio del "campo" (el espacio de soluciones) donde se encuentran las abejas e informar sobre nuevos puntos de alimento.

Estas fases aportan una progresión casi constante al algoritmo, mientras que las abejas scouts se centran en aportar una variación para que nuestro algoritmo no se quede en un máximo local.

En el siguiente pseudocódigo, podemos encontrar la posición y el orden de las fases que hemos mencionado anteriormente. Ilustrando de esta manera el hecho de que las 3 fases son secuenciales y van dentro de un bucle, este bucle está controlado por el parámetro *NCycles* que implicará cuántos ciclos de mejora tendrá el algoritmo.
Este NCycles sería un similar al número de jornadas que realizarían las abejas.

```
initialization();

while NCycles not reached do
        employee_phase();
        onlooker_phase();
        scout_phase()
end

return best_solution();
```

Pseudocódigo simplificado de las fases del algoritmo

Como ya hemos explicado, estamos frente a un algoritmo de enjambre y los agentes que participan en el siguen principios y comportamientos simples presentes en la naturaleza para encontrar una buena solución (De una manera compleja) al problema planteado, es probable que no se encuentre la solución óptima en muchas ocasiones, sin embargo nos encontraremos con una aproximación bastante acertada a dicha solución óptima.
Más adelante explicaremos qué información y qué ítems poseen los agentes para revelar su solución y "comunicarse" entre ellos para determinar qué solución posee una mejor puntuación así como los parámetros que podemos introducir a este algoritmo y que provocarán una variación en el comportamiento.

¿Qué debemos esperar a la salida de nuestro algoritmo? Es una buena pregunta debido a que para el trabajo que se va a realizar se han incluido métricas que normalmente en "producción" no tendríamos por qué probar, pero en resumen recogeremos los tiempos de ejecución del algoritmo y métricas como el mejor score del conjunto de abejas en cada iteración así como la media de los scores del conjunto de agentes por ejemplo.

3 Adaptación del algoritmo ABC al problema de la detección de comunidades

3.1 Adaptando el espacio de red social al algoritmo ABC

En esta subsección describiremos el adaptamiento de los datos que han sido proporcionados para la elaboración del software y cómo dichos datos han sido representados a lo largo del programa.

3.1.1 Representación de las soluciones

Para trabajar con los datos que meteremos debemos idear un método para expresar nuestra solución lo más clara y eficientemente. En un principio nuestro dataset será tratado y expresado como un **grafo G (V, E)** de manera inicial, como se menciona en el apartado **2.1.1** Esta información debemos pasarla a un formato que represente una solución expresando las relaciones entre el grafo, los nodos y las comunidades a las que pertenece. Esta representación de la solución se puede encontrar en [4]

Por lo tanto se ha representado el grafo de entrada al algoritmo como un vector con las siguientes características:

- El **índice** que ocupa el dato corresponde al **identificador del nodo**, siendo el primer dato del vector una referencia (o información sobre) el nodo 0, así con un solo vector podemos almacenar dos datos, la característica que queremos almacenar y el identificador del nodo al que le corresponde esa característica.
- El dato almacenado en el vector es la comunidad o el *cluster* a la que pertenece ese nodo

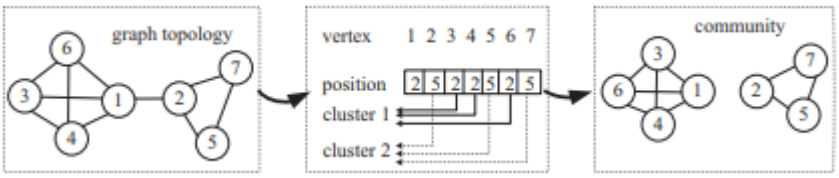

Figura 3-1: Representación Esquemática

De la forma anteriormente expuesta conseguimos expresar la información que nosotros necesitamos con un solo vector de valores, no obstante seguiremos necesitando la estructura del grafo **G(V, E)** que almacena aspectos y características de la red social que vamos a necesitar, sin ir más lejos: Los vecinos de un nodo.

Si combinamos el array de información donde almacenaremos las comunidades con la estructura del grafo **G(V, E)** podremos obtener **funcionalidades como conseguir las comunidades a las que pertenecen los vecinos** de un nodo. Información útil a la hora de inicializar el algoritmo en sí y dar una solución semi-aleatoria pero con influencia de los vecinos de un nodo. Este punto será tratado próximamente más en detalle.

Una vez tenemos esta representación de nuestros factores asimilada, veremos próximamente que al obtener esta información ya seríamos capaces de esbozar una respuesta al problema de la detección de comunidades debido a que sólo necesitamos clasificar cada nodo dentro de una comunidad correspondiente siguiendo una serie de criterios anteriormente mencionados.
No obstante en el siguiente apartado vamos a observar que para optimizar el funcionamiento del algoritmo a la hora de implementarlo y hacerlo funcionar en máquinas físicas nos compensará obtener una serie de factores para aliviar el coste de procesamiento de ciertos puntos importantes del algoritmo.

3.1.2 Agentes

En una primera versión del algoritmo desarrollado, una solución anteriormente mencionada debía ser asignada a una abeja, de este modo, una abeja (o agente) correspondería a una solución del problema única y exclusivamente, debido a que nuestra intención es diferenciar las comunidades de manera que cumplan los requisitos explicados en el apartado de la detección de comunidades.

Esta primera versión fué implementada correctamente y con completa funcionalidad, pero no estando satisfechos con los resultados de las pruebas con respecto al tiempo de computación se decidió realizar cambios en el algoritmo donde cada agente o abeja transportase los siguientes datos:

- **Vector de comunidades:** Siguiendo la representación de [4] explicada anteriormente y siendo el único dato de los agentes en una primera versión en este vector se representa la solución intrínseca del problema

- **Diccionario de comunidades:** Una representación más indexada de las comunidades presentes en la solución anterior que a la hora de obtener ciertos datos nos facilitaría la expresión del algoritmo y nos ahorraría una buena parte del tiempo de cómputo, se trata de un **diccionario clásico (Key, value)** siendo la key la comunidad que queremos explorar y el valor un vector de ids de nodos.

- **Score de la solución:** Donde almacena el score de detección de comunidades anteriormente mencionado, ahorrando la necesidad de recalcularlo cada vez que debamos compararlo con una solución competidora.

- **Número de comunidades:** Un número total de comunidades distintas que posee la solución.

Con esta segunda versión con los agentes teniendo una información más amplia recae más directamente en los agentes la decisión de quedarse con su solución o abandonarla por una mejor al obtener mejor puntuación.

Con esta representación de los agentes obtenemos las ventajas de **acercarnos más al principio fundamental de los algoritmos de enjambre**, donde todos los agentes (en nuestro caso abejas) se comportan de una manera individualista y simple para formar un

algoritmo más complejo al juntarse e interaccionar con los mismos comportamientos sencillos por parte de otras abejas.
También ahorramos bastante tiempo de computación aunque posee la desventaja de usar más memoria a la hora del procesado al almacenar más parámetros, por lo tanto podría influenciar en la definición de parámetros al variar el número de abejas. Y no menos importante, la legibilidad del código empleado y su comprensión se facilita enormemente con la nueva representación de los agentes.

3.1.3 Inicialización del espacio de comunidades

Antes de comenzar a activar los agentes para que realicen sus correspondientes funciones como ya se ha visto en apartados anteriores debemos comenzar en un espacio de soluciones y por supuesto debemos pensar cómo queremos inicializarlo para que sea un espacio "justo" con el que empezar.

Al desconocer por completo la solución óptima no tenemos una manera demasiado lógica que nos permita inicializar nuestro espacio de soluciones de una manera que nos acerque a la verdadera solución a priori, por lo tanto se opta por una inicialización semi-aleatoria de la que partiremos para que mediante las elecciones de nuestras abejas nos acerquen a mejorar el score de la solución.

Comenzamos por inicializar el vector del que hablamos en el **apartado 3.1.1** a un array de longitud **N** (Siendo **N el número total de nodos/usuarios de nuestra red** expresada en grafo) con todos los valores de la comunidad a 0 (La no-comunidad).

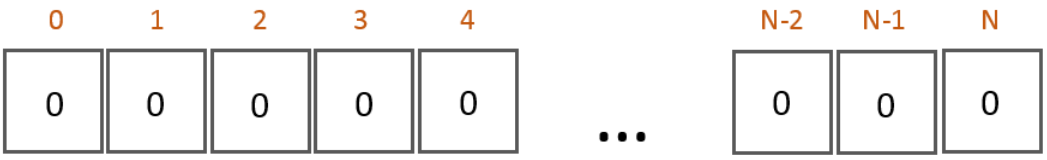

Figura 3-2: Vector inicial de comunidades

Una vez obtenemos este vector recorremos los nodos uno a uno obteniendo las comunidades **asignadas a los vecinos**, con este listado de comunidades desplegamos un abanico de posibilidades amplio para unirse a las comunidades vecinas, pero también añadimos a este listado un identificador de comunidad nuevo, **proporcionando así al nodo la posibilidad de formar una nueva comunidad por su parte**.
Una vez ofrecemos ese abanico de posibilidades al nodo escogemos una comunidad al azar con una elección aleatoria dentro de las posibilidades lógicas que se le ofrecen al nodo.

Una vez este proceso ha acabado obtenemos un vector de comunidades similar al descrito en el apartado 3.1.1 con el que podremos trabajar y el que las abejas podrán explorar a su antojo como se explicará en los siguientes apartados.

3.2 Algoritmo ABC en detalle.

Este algoritmo es el pseudocódigo extraído del paper [5] donde se orienta el funcionamiento del algoritmo ABC para la optimización de ataques a la red, que es otra de las utilidades que hemos mencionado que nos podían proporcionar el estudio de las redes sociales.

En esta aplicación se usa el algoritmo ABC para detectar un punto de la red con una buena **BC (Betweenness Centrality)** Esto es, que es un punto que si fuese eliminado una buena parte de la red quedaría incomunicada o desconectada de otra parte de la red, formando un componente conexo diferenciado del inicial. Esta práctica desemboca en utilidades y motivaciones como cortar la expansión de fake-news, pandemias e información que no se desea que sea esparcida por un lugar concreto de la red.

Este paper ha tenido un gran papel en el desarrollo de nuestro algoritmo debido a que nuestro pseudocódigo es muy similar al suyo, por lo tanto ha servido de guía para la implementación del mismo.

Para comenzar nuestro algoritmo debemos realizar una inicialización de comunidades, en el pseudocódigo de ejemplo se habla de un algoritmo tipo ***avaricioso*** o ***greedy*** que consiste en abarcar el mayor número de nodos posible por una comunidad. Esta inicialización ha sido cambiada debido a que para nuestro problema conviene ofrecer un espacio aleatorio de inicialización para evitar caer en máximos locales de manera sucesiva en cuanto al score de la solución.

Por lo tanto la línea ***"S_1 ← SBA(G, k):"*** será en lugar de SBA el algoritmo de **inicialización descrito en el subapartado anterior,** en el pseudocódigo propio del algoritmo a tratar se traducirá como ***First_Initialization***

Posteriormente como se puede observar pasa a inicializar cada uno de los agentes (abejas) donde obtendrá la mejor solución hasta el momento (descrito con ***Best-so-far-Solution***) y la mejora usando la función ***Generate-Neighbouring*** la cual explicaremos más adelante en detalle debido a que es una de las funciones más clave para el desarrollo del algoritmo a presentar para el trabajo.

Los parámetros de entrada del algoritmo serán el grafo **G** que hemos generado a partir de un input el cual mencionaremos y expondremos más adelante, así como 2 parámetros que iremos variando en el apartado **4.2** (Definición de parámetros).
Estos parámetros son número que corresponden al número de abejas que estarán en nuestro algoritmo (Bees) o el número de ciclos/jornadas que realizarán las abejas.

```
Input: G, Cycles, Bees
Output: Sbest

// Fase de inicialización
S1 ← First_Initialization(G)

for i = 2 to Bees do
        Sbest ← Best_So_Far_Solution();
        Si ← Generate_Neighbouring(G, Sbest);
end

while Cycles not reached do
        // Fase de abejas obreras
        for i = 1 to Bees do
                E ← Generate_Neighbouring(G, Si);
                if Cscore(E) > Cscore(Si) then
                        Si ← E;
                end
        end

        // Fase de abejas observadoras
        for i = 1 to Bees do
                Sj ← Binary_Tournament(S1, S2 … SN);
                O ← Generate_Neighbouring(G, Sj);
                if Cscore(E) > Cscore(Si) then
                        Si ← O;
                end
        end

        // Fase de abejas scouts
        for i = 1 to Bees do
                if Si has not change for limit_iterations then
                        Sbest ← Best_So_Far_Solution();
                        Si ← Generate_Neighbouring(G, Sbest);
                end
        end
end

Sbest ← Best_So_Far_Solution();
```

Pseudocódigo del algoritmo desarrollado

Como se puede observar a simple vista el cuerpo del algoritmo está dividido en las intervenciones de sus correspondientes abejas, tal y como se describe en el apartado **2.3**, cada tipo de agente tiene su fase y en cada una de ellas se repite la función **Generate-Neighbouring** que hemos mencionado.

En el documento original se hace alusión a la función BC, la cual implica en este caso que es una función de fitness basada en la *Betweenness Centrality* de un nodo. En este caso esa misma función ha sido sustituida por ***Cscore*** que hace referencia a ***Community_Score*** , función descrita con anterioridad en la sección 2.1.5, como recordaremos esta función generaba un score que medía cómo de buena es una clasificación de que un nodo perteneciese a una comunidad u a otra. Esta será nuestra función de "fitness" para la mejora de nuestra red.

Por último en la penúltima línea tras la iteración del bucle donde se mejora el espacio de soluciones, se escoge la mejor de todas las soluciones hasta el momento.

3.2.1 Generación de nueva solución

En el apartado anterior hemos hablado sobre una función llamada ***Generate-Neighbouring***, esta función es una de las más primordiales de nuestro software debido a que en ella recae la posibilidad de encontrar una solución mejor que la anterior.

Como hemos podido observar en el pseudocódigo obtiene como parámetros el grafo de la red social y una solución de la cual partimos para generar una nueva. A parte de estos parámetros mete unos rangos que establecen el número de cambios que se realizan.

En nuestro caso esto no varía, pasamos el grafo y nuestra solución como parámetro además del diccionario de comunidades. Para crear una solución nueva se escoge aleatoriamente un único punto de cambio, se detectan los vecinos del nodo que deseamos variar y obtenemos las comunidades a las que pertenecen los mismos.

Una vez tenemos esta lista de comunidades vecinas añadimos la opción de crear una comunidad por su cuenta y ejecutamos una elección aleatoria, almacenando la nueva solución y calculando su diccionario de comunidades así como su score una vez se haya cambiado el punto señalado.

4 Fase de Experimentación

4.1 Descripción del dataset seleccionado

Se ha seleccionado un set de datos con un modelo estandarizado de representación de grafos basada en redes ego encontrado en [4] que se basa en datos anónimos de usuarios de Facebook, este dataset posee 10 redes Ego, muchas de ellas relacionadas entre sí.
En este dataset podemos trabajar como una versión reducida de la red social Facebook o bien podemos trabajar con las redes Ego de manera independiente, de manera conjunta esta red posee 4039 nodos, y 88234 conexiones (Distribuido en 10 ego-networks).

El dataset se compone de varios ficheros correspondientes a cada una de las redes Ego, a continuación explicaremos el contenido de cada uno de esos ficheros siendo la letra X el identificador del nodo al que corresponden estos ficheros descriptivos:

- **X.edges**: Determina las conexiones del grafo. Cada línea es una conexión y tenemos los dos identificadores de los nodos que están conectados de manera que en este caso no se distingue entre emisor y receptor, sólo conexiones bilaterales. Estas conexiones son entre los *Alters* de la red y no se incluye la conexión con el ego debido a que por el concepto de la red ego, todos van conectados a él.

- **X.egofeat**: Este archivo contiene un único vector binario que indica la presencia (1) o ausencia (0) de una determinada característica en el perfil del ego. El significado de cada posición aparece en el fichero de *X.featnames*.

- **X.feat**: En este archivo te ponen las características de cada uno de los *Alters*. en este caso existe, una línea por alter, el primer número se corresponde con el Id del alter, y luego viene el vector binario con las características que ha especificado.

- **X.featnames**: y aquí está el archivo que contiene el significado de cada uno de los bits del vector de características.

A continuación se expone una tabla con la información estructural de cada una de las redes Ego en la que se especificarán el número de nodos y de conexiones entre los mismos.

Ego (Identificador)	Número de nodos	Número de aristas
0	348	2519
107	1046	26749
348	228	3192
414	160	1693
686	171	1656
698	67	270
1684	793	14024
1912	756	30025
3437	548	4813
3980	60	146

Tabla 1: Listado de ego-networks y características

El output de nuestro programa será parametrizado de la siguiente manera:

- **Solución Original:** Vector o array de comunidades descrito en el apartado **3.1.1** donde se devuelve el mejor resultado de todos una vez se ha identificado y su score.

- **Solución con Métricas:** Estas soluciones no aportan valor real para la finalidad del algoritmo pero nos permiten medir su efectividad y discutir los resultados. Se componen de:
    - **Listado de mejores resultados** (best score) tras cada generación para observar el progreso del algoritmo
    - **Listado de resultados medios** (average score) tras cada generación entre todos los obtenidos por los agentes para poder determinar el progreso del algoritmo. Incluye la varianza.
    - **Listado de tiempo de ejecución** para determinar el tiempo que emplea nuestro algoritmo con objetivo de ver los cambios en el tiempo de ejecución al variar los parámetros.

4.2 Definición de Parámetros

Ya hemos tratado anteriormente la complejidad de la cual es la naturaleza del algoritmo. Esta naturaleza NP-Hard, nos lleva a emplear metodologías típicamente heurísticas a la hora de probar nuestro software.
No obstante, antes de probar el funcionamiento de ABC con un extracto directamente de Facebook, de manera que se esté simulando una red social, debemos definir los parámetros óptimos para esta prueba.

Como se menciona en el apartado anterior, tenemos un **dataset con 10 redes Ego** distintas de las cuales hemos escogido la que posee la media de tamaño. Las razones para escoger la red ego del nodo 0 fueron estar en un punto donde no es una toma muy pequeña ni muy grande.

Siendo una toma muy pequeña perdemos la credibilidad a la hora de fijar parámetros debido a que un salto generado por las abejas scouts adquiere demasiada importancia. En cambio, tomando una red muy grande se eleva considerablemente el tiempo de computación, haciendo mucho menos viable variar los parámetros como prueba por el tiempo de computación.

Más específicamente nuestra tarea actual consistirá en fijar **2 parámetros** principales para el desarrollo de la "colmena" como son:

- **Número de ciclos** que se repetirá en la colmena, **similar a las generaciones en un algoritmo genético** con la diferencia de que las abejas cumplen "jornadas" de trabajo, no son individuos que mueren para dar paso a otros mejores

- **Número de abejas** de la colmena que están realizando esos ciclos. Al variar el número de **agentes** que participan en el algoritmo podemos esperar cambios más ágiles por cada ciclo.

Para realizar una correcta fijación de parámetros hemos probado de manera inicial los siguientes valores para estudiar sus resultados y detectar los valores más eficaces entre ellos:

- **Número de ciclos**: 10, 30, 50, 100, 200, 500, 1000
- **Número de Abejas:** 5, 10, 20, 40
- **Número de Repeticiones por configuración**: 5

Como resultados a mostrar realizaremos 3 gráficas explicativas con los valores de la mejor puntuación, la media entre las puntuaciones y el tiempo por cada ciclo o generación que transcurre.
Como podremos ver en el **anexo A** las gráficas son muy ilustrativas. En este apartado sólo comentaremos las gráficas más relevantes a la hora de la definición de los parámetros y las tendencias entre todas las gráficas.

Hasta llegar al número de ciclos 200 todas las gráficas, en mayor o menor rectitud de sus líneas, siguen una forma similar a la expuesta a continuación.

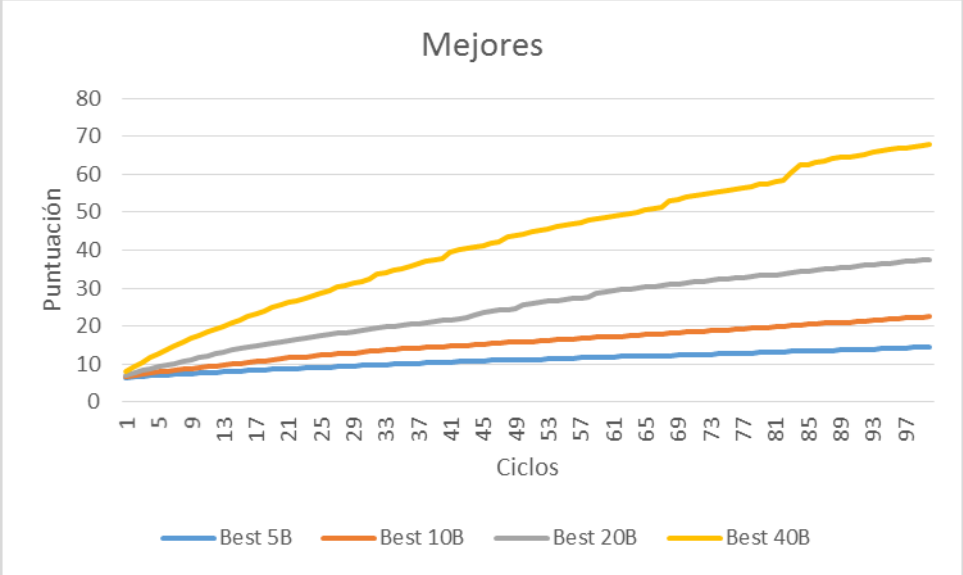

Figura 4-1: Gráfica de mejores puntuaciones en 100 ciclos

Como podemos observar existe una grandísima diferencia en cuanto a la puntuación si variamos el número de abejas entre 5 y 40. Dejando a un lado esta diferencia podemos observar que estas gráficas poseen formas similares con la diferencia de otorgar valores más altos.

Hasta el momento parece la secuencia habitual que debería tener un algoritmo heurístico, una constante pero lenta mejoría que incrementa considerablemente cuando se incrementan el número de agentes, provocando así una mejoría más temprana. No obstante esto varía cuando dejamos tiempo para que las abejas scouts tengan ocasión de actuar como veremos en la siguiente gráfica con 200 generaciones.

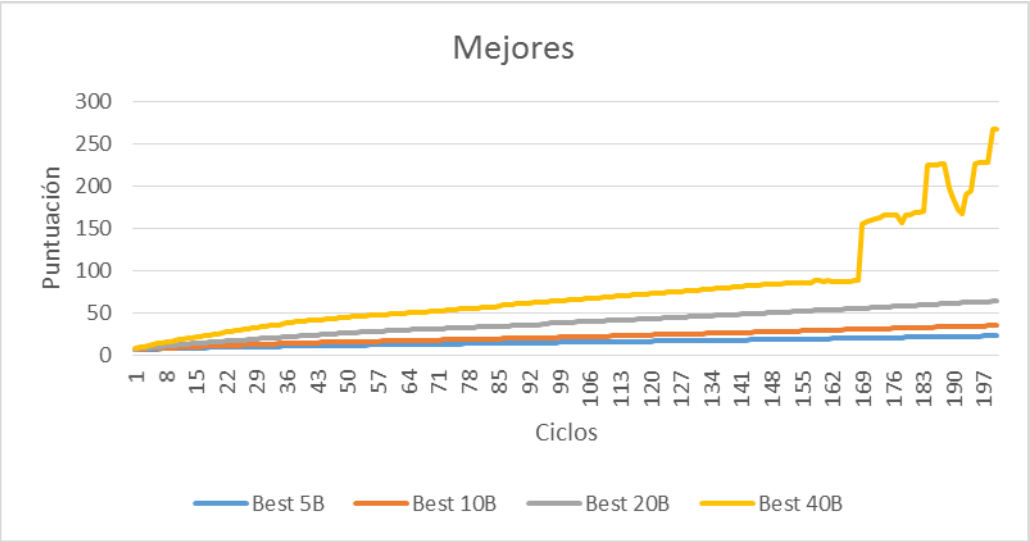

Figura 4-2: Gráfica de mejores puntuaciones en 200 ciclos

Como podemos observar en esta gráfica con **200 generaciones** existe lo que podríamos percibir como una irregularidad **entre los ciclos 162 y 169** que hace que **la mejor puntuación se dispare** para la línea de ejecución con 40 abejas, haciendo su avance ascendente y más irregular que como habíamos visto previamente en el algoritmo.

Este fenómeno se debe a la intervención de las abejas scouts en el ciclo de vida del algoritmo. Hasta ahora latentes en su mayoría, estas abejas aportan variedad y aleatoriedad a nuestro algoritmo heurístico, provocando que, como es el caso, nuestro algoritmo no se quede rondando un máximo local.
Cuando las abejas *employee* o las abejas *onlooker* **no son capaces de encontrar una mejora local** intervienen las abejas scouts, haciendo variar de una manera más radical la solución que lleva una abeja que se ha quedado atascada en un **máximo local.**

Esto puede provocar irregularidades en la gráfica al entrar en efecto, haciendo cambiar la continuidad ascendente gradual de nuestros resultados hasta el momento.
Si vemos las gráficas de la media de soluciones, hasta estas irregularidades eran más suaves y constantes que la gráfica de mejores resultados, aun así cuando este fenómeno ocurre la media puede llegar a desplomarse para luego ascender de una manera muy ágil otra vez como podemos apreciar a continuación.

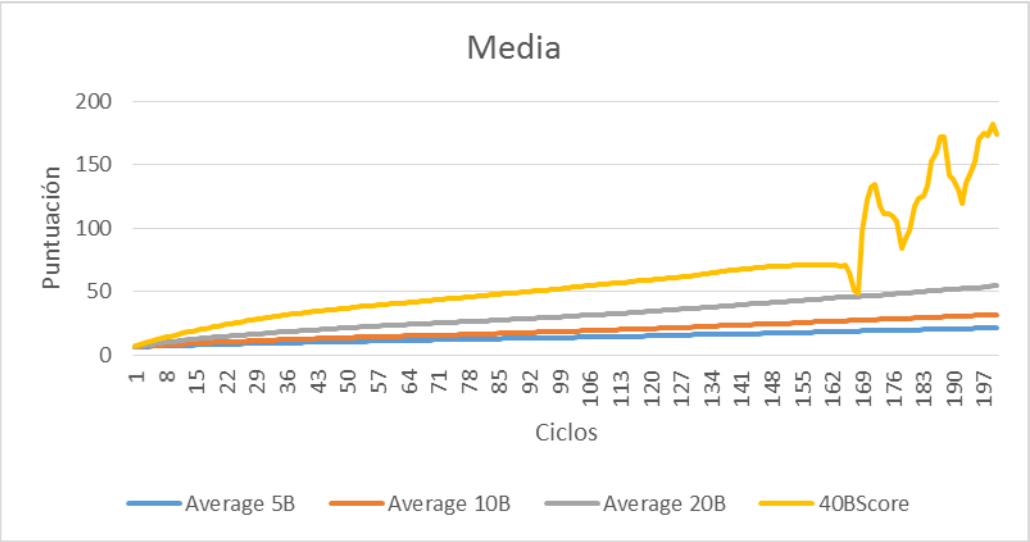

Figura 4-3: Gráfica de media de puntuaciones en 200 ciclos

A lo largo de esta serie de gráficas, así como las del anexo A podremos observar que la línea de 40 abejas sobresale de manera muy evidente frente al resto de fonfiguraciones, por otro lado como veremos a continuación será la configuración que más tiempo consuma a lo largo de la ejecución.

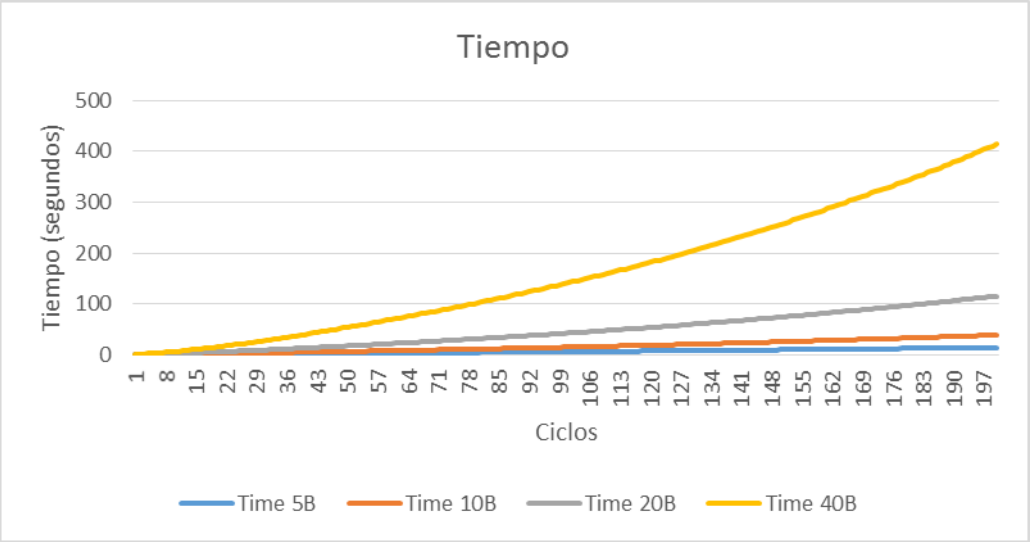

Figura 4-4: Gráfica de tiempo de ejecución en 200 ciclos

Como podemos apreciar en la anterior figura, la gráfica de tiempo con **40 abejas** va adquiriendo poco a poco la forma propia de una progresión exponencial, la cual veremos más fácilmente en el **Anexo A** o en la siguiente gráfica de tiempo.

La prueba con mayores parámetros fue a la cual se le asignó **1000 ciclos** de cómputo, adquiriendo unos muy buenos resultados que acabaron convergiendo en una puntuación concreta como se muestra a continuación.

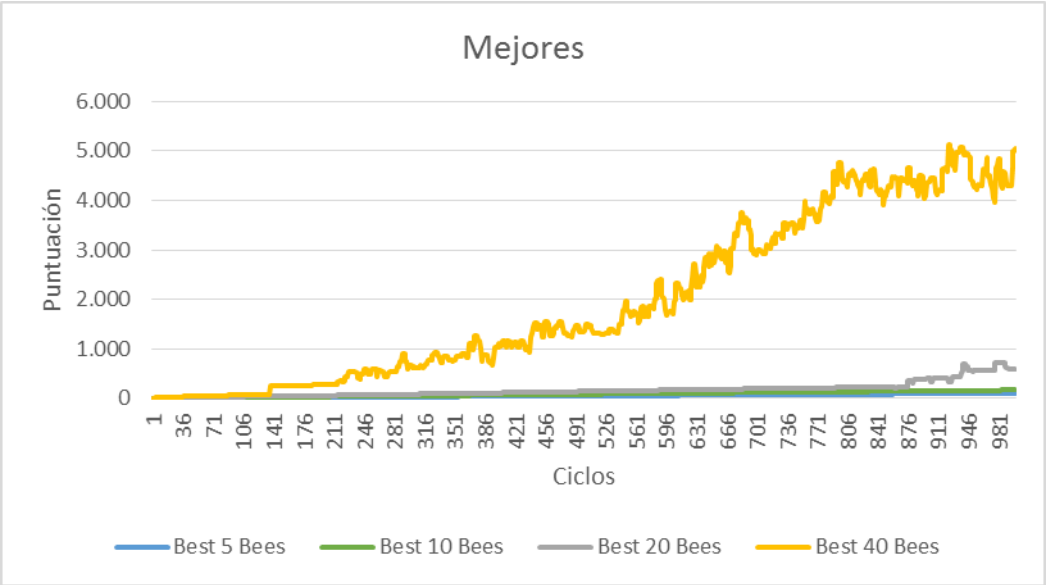

Figura 4-5: Gráfica de mejores puntuaciones en 1000 ciclos

A partir aproximadamente del ciclo **800 los mejores resultados parecen haber alcanzado un máximo local**, a continuación verificaremos la mejora y la convergencia de este gráfico con la media del score de todas sus abejas.

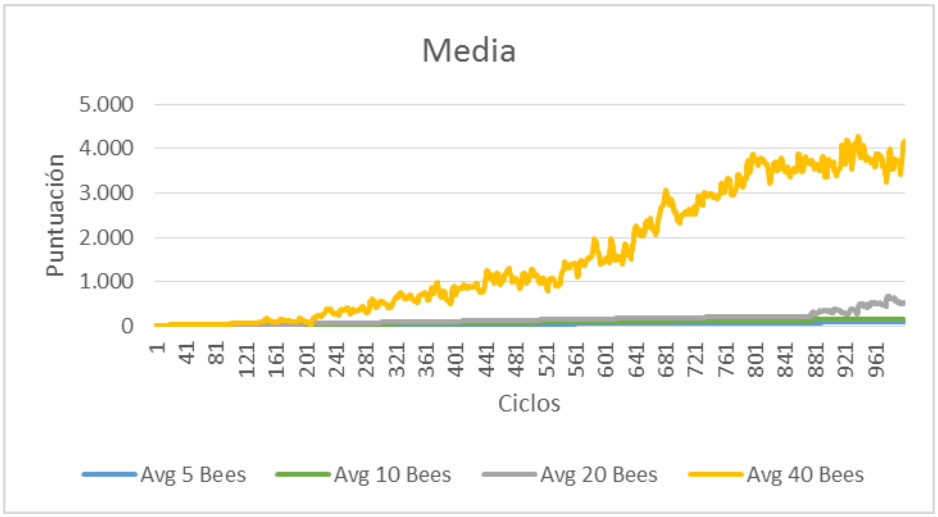

Figura 4-6: Gráfica de la media de puntuaciones en 1000 ciclos

Como podemos observar en estos dos gráficos la secuencia de puntuaciones de **20 abejas comienza en el ciclo 800 a incrementarse de igual manera que 40 abejas**, este fenómeno lo exploraremos al fijar el siguiente parámetro: el número de agentes o abejas con los que ejecutamos el programa. Pero antes aclararemos el tiempo de ejecución:

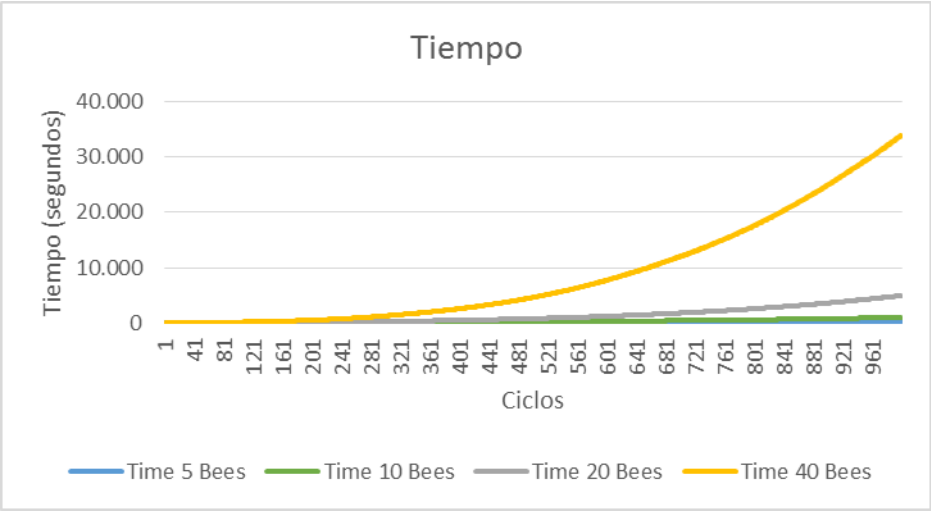

Figura 4-7: Gráfica de tiempo de ejecución en 1000 ciclos

Como podemos observar en la gráfica de tiempos de ejecución anterior, 1000 ciclos con 40 abejas equivalen a **35.000 segundos** por cada una de las repeticiones que realiza el algoritmo. Esta cifra equivale a aproximadamente a **10 horas de ejecución** por cada una de las repeticiones.
En caso de realizar las pruebas reales con 5 repeticiones al igual que la definición de parámetros, nos encontraríamos con 50 horas de media para obtener los resultados de una sola de las redes ego que se desean analizar.

Por esta razón de limitación de tiempos junto a la convergencia de los datos (a partir de cierto punto no mejora mucho) y las los límites para la entrega del trabajo se fija el parámetro de ciclos a 500, donde podremos encontrar una gráfica en crecimiento como la siguiente.

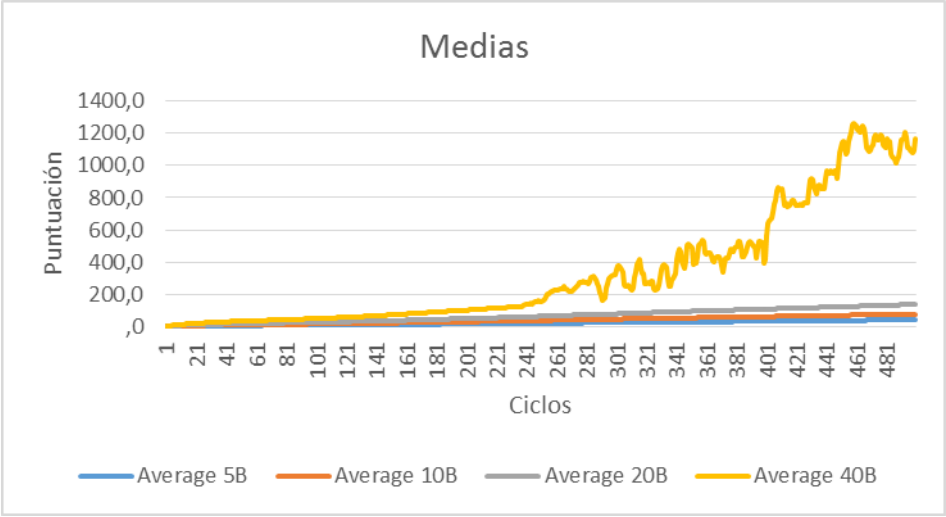

Figura 4-8: Gráfica de media de puntuación en 500 ciclos

Por otro lado compensaremos la demora que existe a la hora de empezar a mejorar el score de manera notable como veremos en el siguiente subapartado.

4.2.1 Parámetro de número de abejas (pruebas extra)

Tras las pruebas con la variación de los ciclos de trabajo de las abejas, se observó que el número de abejas era tremendamente importante en la realización de las pruebas. Cuanto más se aumenta este parámetro antes se comienza a obtener una solución más acertada a sus predecesoras en cuanto a lo que ciclos se refiere.

Este fenómeno se conoce como **exploración**, que en el ámbito de la inteligencia computacional se refiere a encontrar soluciones mejores más rápidamente extendiendo el rango de búsqueda.
Su opuesto es la **explotación**, que como hemos hecho hasta ahora intenta mejorar de manera más local una solución realizando una búsqueda más exhaustiva y por lo tanto costosa, sería algo así como un refinamiento de una solución válida para hacerla mejor.

Al incrementar el número de abejas en una gráfica aumenta tanto su tiempo de ejecución como cuando se empieza a mejorar de una manera significativa, aunque mucho más irregular y abrupta. En este caso al tener un margen de mejora muy elevado nos interesa obtener los resultados más altos empleando la menor fuente de recursos obteniendo así un buen rendimiento.

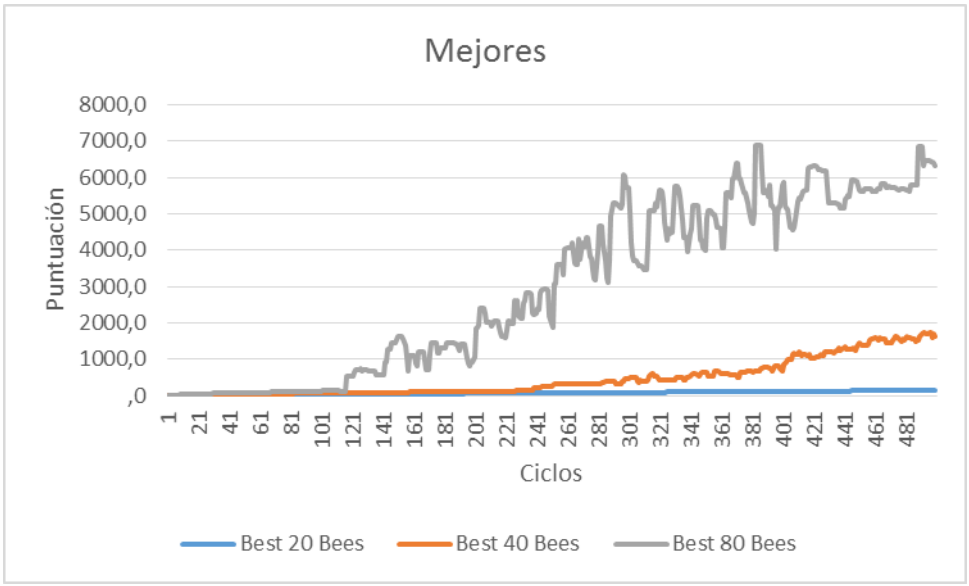

Figura 4-9: Gráfica de comparación de puntuación respecto a número de abejas

Fijando el parámetro de ciclos en 500 (Para obtener mejor tiempo de ejecución) hemos introducido al algoritmo que compute con 20, 40 y 80 abejas.
Como era de esperar los resultados con 80 abejas están muy por encima de 40 por lo cual se escogerá el parámetro de 80 abejas para la prueba real y así obtener un resultado más elevado en menor número de ciclos.

Si comparamos la puntuación del algoritmo con el equivalente en 1000 ciclos y 40 abejas las puntuaciones son significativamente más elevadas. En la mitad de ciclos se logra un score medio de 6000 rozando en ocasiones los 7000, mientras que con 40 abejas parece la mejor puntuación oscila entre los 4000.

En cuanto a lo que nos referimos a tiempo es más o menos equivalente doblar el número de ciclos con doblar el número de abejas como se puede observar en la comparativa de gráficas siguiente:

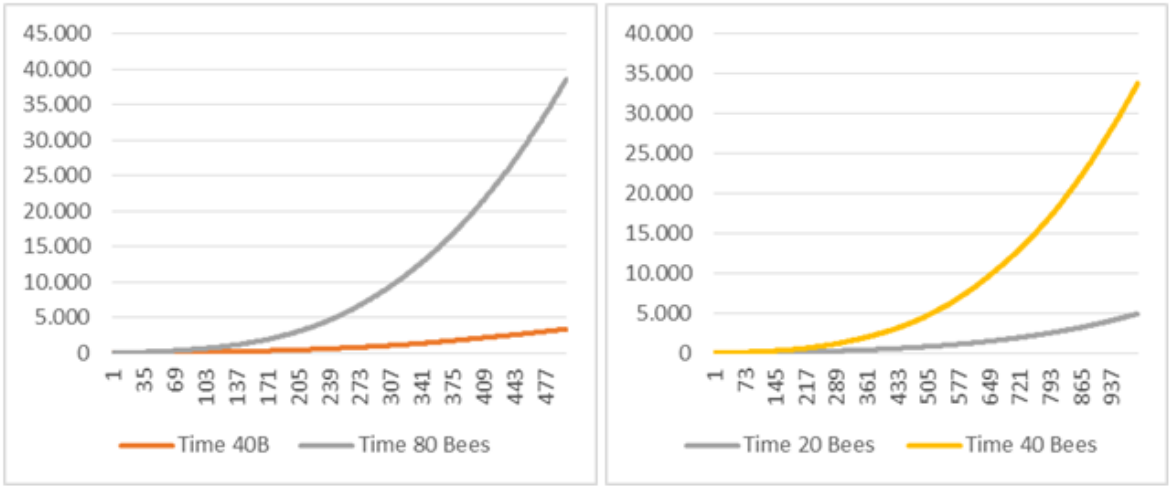

Figura 4-10: Comparación de tiempo entre 500 ciclos y 40 y 80 abejas contra 1000 ciclos con 20 y 40 abejas

Como se observa, el tiempo de **500 ciclos con 80 abejas ronda los 40000 segundos**, cuando **1000 ciclos con la mitad** de abejas no nos proporcionan mejores resultados y disminuyen solo unos 5000 segundos.

Para la prueba real se escogerán los parámetros de 500 ciclos y 80 abejas a la hora de ejecutarlo por la demostración de eficacia a la vista de los resultados obtenidos.

4.3 Prueba Real

A continuación en este apartado se va a exponer la información que hemos extraído de los resultados que hemos obtenido al ejecutar el algoritmo para varias redes ego de mayor magnitud que con la que se han realizado las pruebas, así como para toda la red de Facebook que poseíamos de prueba mencionada en el dataset del apartado 4.1.

En la siguiente tabla podemos encontrar los resultados obtenidos para las redes que se han probado y por último expondremos los resultados obtenidos para la red completa.

Debemos recordar que estos resultados tienen como parámetros que se reproducirán 3 veces (3 repeticiones para obtener la media de los resultados) y se ejecutarán con 500 ciclos y 80 abejas por repetición.

Ego	Nodos	Aristas	Best (score)	Average (score)	Tiempo (segundos)
0	348	2519	8147.87	7225.99	41620.79
107	1046	26749	1011.28	967.78	56108.64
348	228	3192	13686.89	12538.97	33551.79
414	160	1693	5769.57	5367.94	15273.54
686	171	1656	6885.24	6362.53	16267.63
698	67	270	7590.45	7368.88	11756.98
1684	793	14024	11217.43	7903.27	27604.87
1912	756	30025	4115.31	4086.07	46992.31
3437	548	4813	5700.93	4612.73	23892.01
3980	60	146	2256.32	1985.55	9856.65

Tabla 2: Resultados de pruebas reales sobre las ego-networks

Para la red completa se obtienen los siguientes resultados:

- **Mejor Puntuación**: 272.74
- **Media de Puntuación**: 252.96
- **Tiempo de ejecución empleado**: 75261.04 segundos

En el siguiente apartado podremos discutir y teorizar sobre los resultados obtenidos por la prueba real y las conclusiones obtenidas para el conjunto de pruebas y definición de parámetros.

4.4 Discusión de Resultados

Como hemos podido observar en las ejecuciones anteriores, existe una clara relación entre el tamaño de la red que vamos a procesar y la puntuación máxima que podemos llegar a generar.
Este fenómeno puede deberse a la misma razón por la cual se realizó una segunda tanda de experimentos para definir los parámetros en la sección 4.2. El número de abejas, principalmente, influía en la manera en la que el score de la solución se comenzaba a disparar de una manera abrupta.

Como recordaremos de la sección de definición de parámetros, cuando incrementábamos el número de abejas, lo que provocábamos era que las abejas scouts comenzasen a actuar antes, proporcionando de esta manera una expansión más explosiva de las abejas por el espacio de soluciones. Esta expansión más rápida y eficaz en los inicios de un algoritmo nos proporcionaba saltos inmensos en la puntuación con pocas generaciones.

Otra razón por la cual el score es menor puede deberse al tamaño de la red y que los cambios realizados son menos significativos en cuanto a la puntuación máxima posible, por lo tanto en este caso el cambio de un nodo, aunque esté bien definido, no podría compararse al índice de mejora de una red mucho menor en tamaño.

Por otro lado aunque el índice de mejora sea más lento podemos observar en la siguiente gráfica que la mejora es constante y ascendente.

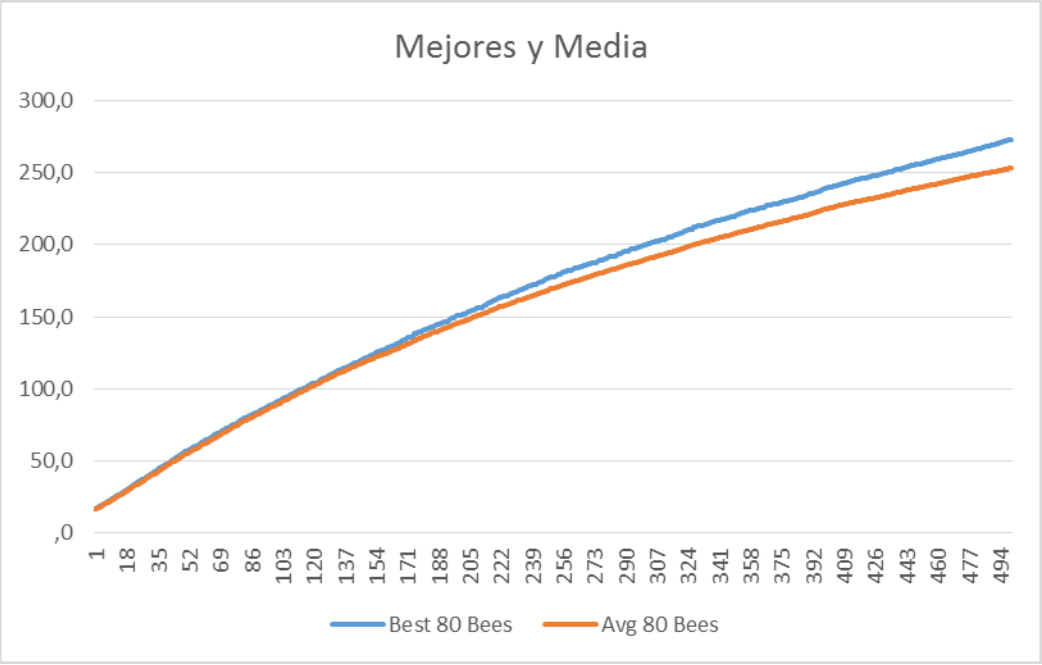

Figura 4-11: Puntuaciones de red completa

Si por otro lado comparamos las puntuaciones de este conjunto de redes con las de la red más grande de entre las 10 que la componen podemos observar que también queda reducido el score que genera en el tiempo asignado, al igual que queda suavizada su gráfica.

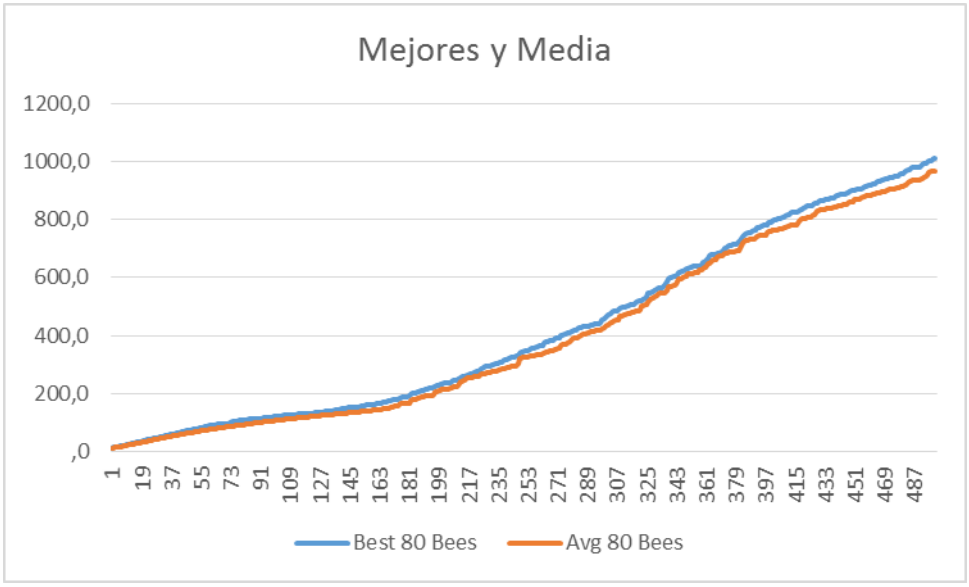

Figura 4-12: Puntuaciones de ego 107

Esta suavidad de la que hablamos es debido a que los cambios de comunidades dentro de una red bastante más grande con la cual hemos realizado las pruebas no son tan significativos.

Teniendo una visión más global de los resultados, una vez se han realizado las pruebas reales, se puede llegar a pensar que posiblemente según se aumente el tamaño de la red debemos incrementar junto al mismo el número de abejas y/o de generaciones para obtener una progresión acorde con los experimentos realizados en el apartado de definición de parámetros.

Por otro lado podemos concluir que el algoritmo es válido para resolver el problema de la detección de comunidades en redes sociales. La progresión en la puntuación indica que es un algoritmo que mejora el score y por lo tanto progresa y no se queda atascado en el problema que se le plantea.

Respecto al tiempo, como veremos a continuación, el tamaño de la red no parece ser un factor excesivamente importante debido a que influye mucho más el número de generaciones o el número de abejas si lo comparamos.

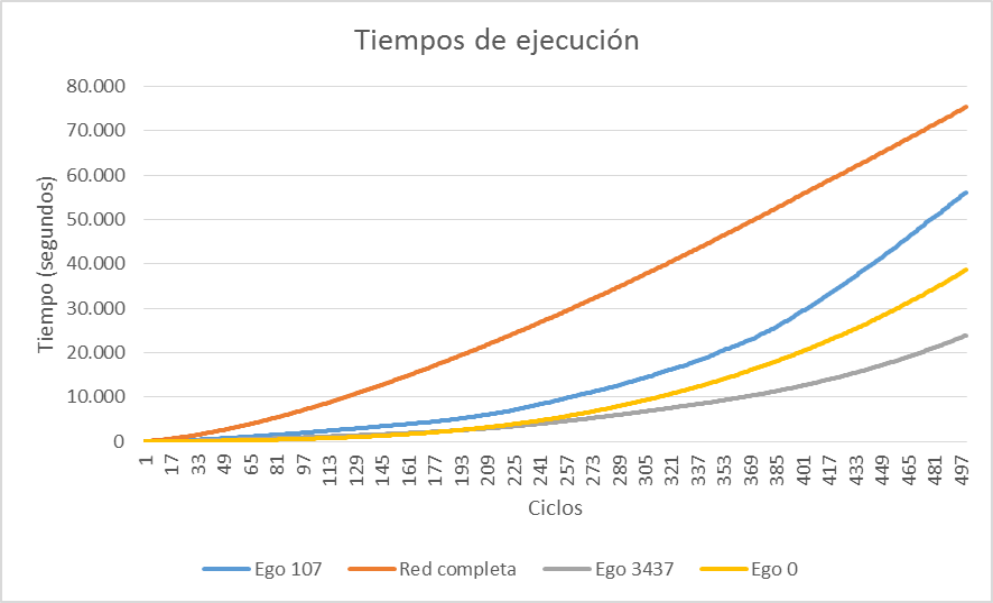

Figura 4-13: Comparativa de tiempos de ejecución

Con motivo de no sobresaturar la gráfica se escoge una de las redes ego más grande (107), así como con la que hemos realizado las pruebas (0) que corresponde con un valor intermedio y otra más pequeña para percatarnos de la relación por tamaño entre las mismas.

Observando la gráfica, el tiempo de ejecución de la red completa es mayor que las redes ego que componen la muestra por separado, pero sin embargo no equivale a la suma aritmética de tiempos, debido a que es mucho menor. También si nos fijamos veremos que la curvatura de la línea de la red completa es mucho menos pronunciada que la de los egos por separado, lo cual nos lleva a creer que el algoritmo se comportará de una manera más suavizada a la hora de subir el número de ciclos.

Como consecuencia podemos concluir que el algoritmo no se ve excesivamente perjudicado por el tamaño de la red que se trata, si no que se deben escoger de una manera acertada sus parámetros. Este algoritmo podría llegar a ser una opción competente a la hora de tratar con el problema de detección de comunidades.

5 Conclusiones y trabajo futuro

5.1 Conclusiones

Este Trabajo de Fin de Grado ha sido concebido para tratar de demostrar que los algoritmos de enjambre, en concreto ***Artificial Bee Colony***, pueden aplicarse para tratar el problema de detección de comunidades.

Entre muchos otros problemas y desafíos que nos proporciona este nuevo mundo de las redes sociales, existen numerosos estudios para lidiar con ellos y entre estos se encuentra el problema seleccionado: La detección de comunidades.

Este problema del que hemos hablado previamente posee un interés en todo el ámbito de las redes sociales que lleva varios años desarrollándose y cobrando cada vez más importancia en nuestras vidas, convirtiéndose en un entorno de referencia entre las personas en cuanto a la comunicación y la manera de difundir y compartir información. De esta manera cada vez se tiene mayor cantidad de datos sobre los que podemos trabajar y obtener resultados dependiendo del problema que se aplique.

Por otro lado, los algoritmos heurísticos han ganado una gran importancia debido a que este avance de las redes no solo se puede percibir en las mismas, si no que cada día que pasa existen más datos y como respuesta, nuevos y mejorados algoritmos que nos permiten definir resultados entre tantísima información.
Estos algoritmos nos permiten obtener soluciones aceptables frente a problemas computacionalmente muy complejos en un tiempo relativamente aceptable. Esto se debe a que ya no es necesario inspeccionar cada una de las partes de un grafo o de un espacio de búsqueda, si no que se verán guiados por una función que describe cómo de buena es una solución, se la asigna el nombre de función fitness o en nuestro caso, score.

De entre todos estos posibles algoritmos se ha escogido ***Artificial Bee Colony***. Se trata de un algoritmo de enjambre inspirado en ciertos comportamientos de las abejas melíferas que ha demostrado previamente poder usarse en otros problemas del tratamiento de redes sociales con unos buenos resultados.

En el Trabajo de Fin de Grado realizado se ha diseñado una implementación de ABC al problema del que hemos hablado, asimilando los conceptos principales del algoritmo en sí y adaptando su funcionamiento, el cual estaba inicialmente enfocado en abordar otro problema como el de *Betweeness Centrality*, de manera que se han rediseñado varios principios de esta solución.

Se ha realizado una fase para definir los parámetros más eficaces con una muestra de información más reducida que la prueba real (solamente una ego-network). De esta manera se han detectado las opciones más efectivas de configuración y se han obtenido distintas

conclusiones expuestas en apartados anteriores con respecto al progreso del algoritmo cuando se variaban los parámetros.

Aunque esta fase de definir los parámetros no entre dentro de la fase real nos ha proporcionado una gran cantidad de resultados y comportamientos útiles para conocer verdaderamente las tendencias de este particular algoritmo.

Las pruebas reales sobre una red de Facebook, tanto como por separado en los distintos egos como la red ego completa, han demostrado que el algoritmo tiene un gran potencial para resolver el problema de la detección de comunidades. Como se ha observado en el Trabajo realizado, podemos obtener un score bastante elevado que corresponde con una buena división de comunidades como deseábamos.

Por otro lado hemos podido percibir también que el tiempo de ejecución no se ve excesivamente afectado al incrementar el tamaño de la red, sin embargo el score con el que tratamos evolucionará de una manera más pausada debido a la extensión de la misma.
Este fenómeno puede deberse a que los cambios que ejercen las abejas no son tan significativos al existir un mayor número de nodos en la red.

5.2 Trabajo futuro

En esta sección se detallan posibles mejoras, cambios e investigaciones planteadas que no se han recogido en el trabajo actual.

5.2.1 Inicialización mejorada

Como ya hemos visto en la sección 3, la manera de asignación de comunidades nueva es completamente aleatoria una vez hemos recogido la lista de vecinos de un nodo para el que queremos definir la comunidad…

Esta asignación es "justa" debido a que interviene la aleatoriedad y se ofrece al nodo la posibilidad de crear una comunidad por su cuenta, instrucción que aporta variedad y ofrece al algoritmo heurístico una vía de escape para poder incrementar el número inicial de comunidades según se necesite. No obstante existe la posibilidad de crear un sistema menos aleatorio que asigne las comunidades en función del número de vecinos que pertenecen a una comunidad X.

Para ver esta explicación de una manera clara expresaremos un ejemplo:

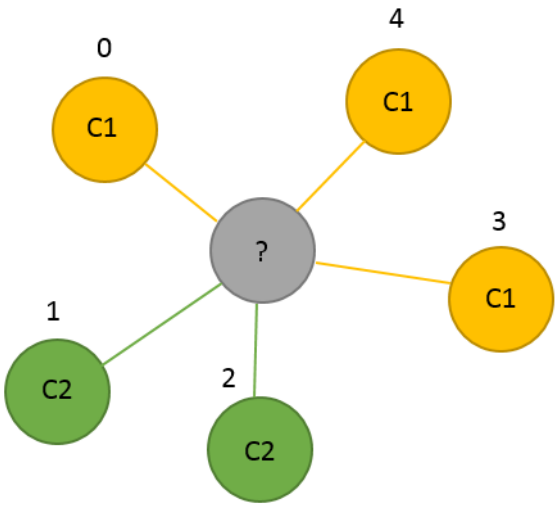

Figura 5-1: Comunidades vecinas a nodo de ejemplo

Como podemos comprobar, al tener 3 vecinos pertenecientes a la **comunidad 1** y 2 pertenecientes a la **comunidad 2**, si incluimos la posibilidad de crear una nueva comunidad por su cuenta tendremos un array similar a este:

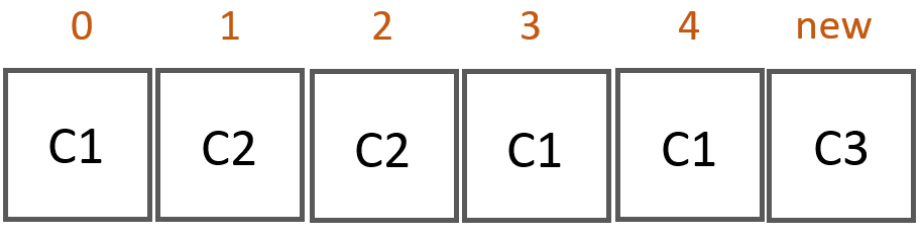

Figura 5-2: Array de posibles valores

Donde las probabilidades si se ejecuta un random que escoja entre uno de los ítems de la lista serían:

$$Prob(C1) = \frac{3}{6} \qquad Prob(C2) = \frac{2}{6} \qquad Prob(C3) = \frac{1}{6}$$

Esta mejora podría mejorar significativamente el punto de partida del algoritmo así como la decisión de la función **Generate-Neighbouring** encargada de obtener una nueva solución y compararla con la solución previa.

La parte negativa es que podría limitar en cierta medida la toma de decisiones y la experimentación para realizar cambios muchas veces necesarios en la red.

5.2.2 Parámetro de k-Cambios

Como un posible cambio con su posibilidad de investigar el comportamiento del algoritmo ante este cambio sería añadir un parámetro k con el que podríamos variar el número de cambios que se producen en una solución propuesta por nuestra función *Generate-Neighbouring*.

Actualmente este parámetro se establece a 1 debido a que en todas las soluciones nuevas generadas se varía exclusivamente un nodo. Este cambio provocaría un avance más rápido de soluciones a la hora de avanzar en las generaciones.

La parte negativa es la incertidumbre que genera, debido a que al meter por ejemplo 4 cambios en una misma variación de la solución la mejora del score será la media entre lo que varía el score con cada uno de los cambios. Si 2 de ellos aportan pero los otros 2 cambian a peor en mayor medida, el cambio no se produce y se pierde ese posible margen de mejora que obtendríamos haciendo los cambios por separado.

En conclusión, se gana agilidad a la hora de mejorar, pero se pierde en gran medida precisión a la hora de realizar los cambios, debido a que cuanto más grande sea K menos posibilidad de realizar un cambio enteramente positivo para el score de la red.

5.2.3 Realización de clusters o patrullas de abejas

Como una posible mejora o variación funcional del algoritmo sobre el que se ha trabajado a lo largo de este proyecto, se plantea la posibilidad de dividir patrullas de abejas para que, de esta forma, cada patrulla posea sus propias abejas scouts y que no paren de trabajar de una manera constante.

De esta manera se consigue una continua exploración aleatoria del campo de soluciones posibles. Las patrullas de abejas van generando constantes cambios aleatorios de zona en la que se varía la solución, no tienen que esperar a que no se pueda mejorar una solución rondando los vecinos a la misma.

El funcionamiento más detallado consiste en crear patrullas de abejas que continuamente busquen cambios beneficiosos para el score, y que mientras las abejas scouts plantean mover el punto donde se encuentra la patrulla, las abejas obreras y observadoras mejoran los vecinos del punto donde se sitúan anteriormente.

Como consecuencia esto podría influir en el algoritmo de manera que se realiza una fase de exploración mucho más intensa, explosiva e irregular que de manera habitual, pero como compensatoria podemos progresar mucho más rápido en la fase temprana en el algoritmo.

Referencias

[1] "Digital 2019: Global Digital Overview", 22 de Febrero de 2019 https://datareportal.com/reports/digital-2019-global-digital-overview

[2] C. E. Borges, J. L. Montaña, "Algoritmos Bioinspirados" 5 de Marzo de 2019, http://paginaspersonales.deusto.es/cruz.borges/Papers/10APIAXXI.pdf

[3] Dervis Karaboga, B. Basturk, "On the performance of artificial bee colony (ABC) algorithm" Applied Soft Computing, Volume 8, pp. 687-679.

[4] Qing Cai, Maoguo Gong, "Greedy discrete particle swarm optimization for large-scale social network clustering", Elsevier Inc, 2014

[5] Manuel Lozano, Carlos García-Martínez, Francisco J. Rodríguez, Humberto M. Trujillo, "Optimizing network attacks by artificial bee colony", Information Sciences, 2017, 30-50

Anexos

A Gráficas en la definición de parámetros (Ciclos)

En el siguiente anexo se dispone a ilustrar mediante gráficas la progresión del algoritmo ***ABC*** según se varía uno de sus parámetros: los ciclos de las abejas.

Como se menciona en el apartado 4.2 se utiliza un clúster de computación presente en la EPS bajo el mando del tutor académico.
Aclarar que si este programa se ejecuta en un ordenador personal de gama media-alta en el año referente al 2019 los tiempos de ejecución y por tanto los presentes en este apartado pueden multiplicarse por un factor de hasta 3 veces más.

También se aclara que los números y valores presentes en las gráficas son medias aritméticas entre los valores devueltos por el algoritmo. Estos valores se obtienen ejecutado un número de repeticiones concreto, ya que sabemos que existen componentes aleatorios en los algoritmos heurísticos pero se intenta definir una directriz común para todas las ejecuciones.
Este parámetro de número de repeticiones ha sido fijado de manera constante a 5, posteriormente se ha realizado la media aritmética de los valores entre las 5 repeticiones y con dichos valores se han realizado las gráficas que se pueden ver a continuación.

Las últimas 3 gráficas se realizan a posteriori para fijar de una manera más amplia el parámetro de número de abejas, debido a que según iban saliendo los resultados se contemplaba que el este parámetro tenía una gran influencia sobre los resultados.

Número de ciclos: 10
Número de Abejas: 5, 10, 20, 40

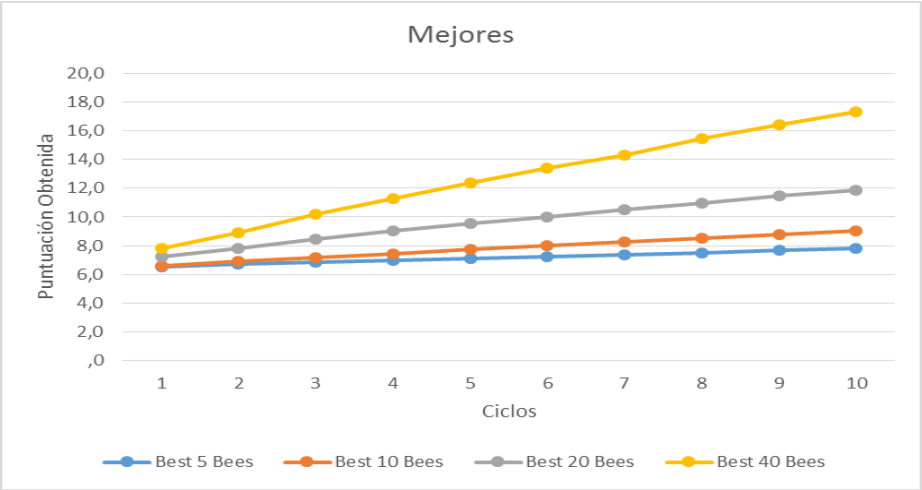

Figura 0-1: Mejores con 10 ciclos

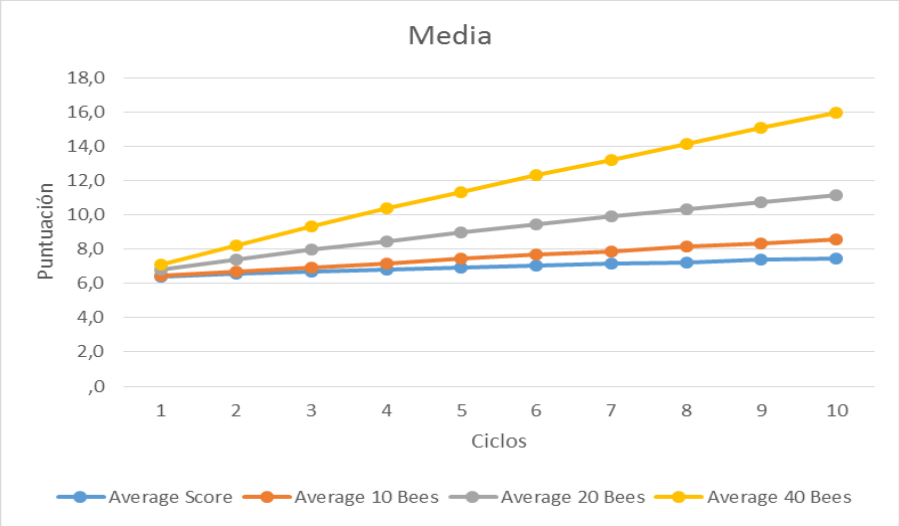

Figura 0-2: Media con 10 ciclos

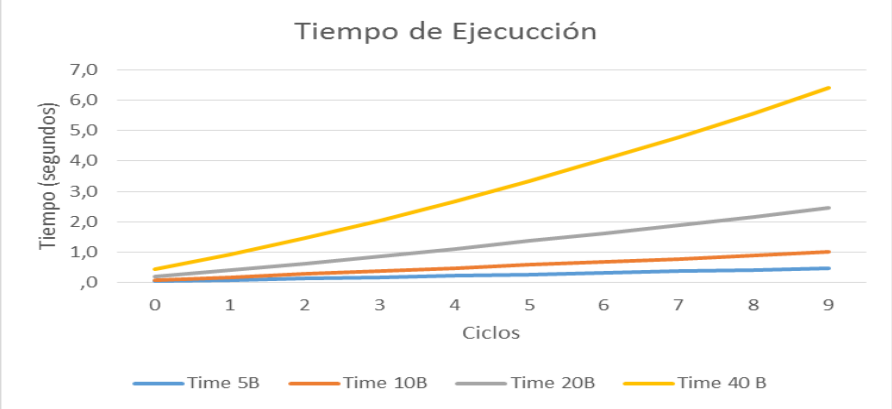

Figura 0-3: Tiempo con 10 ciclos

Número de ciclos: 30
Número de Abejas: 5, 10, 20, 40

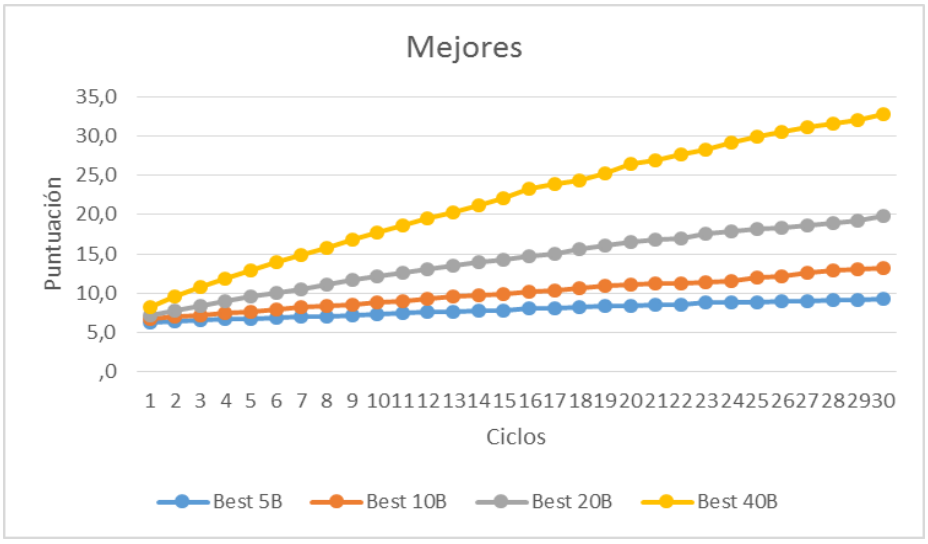

Figura 0-4: Mejores con 30 ciclos

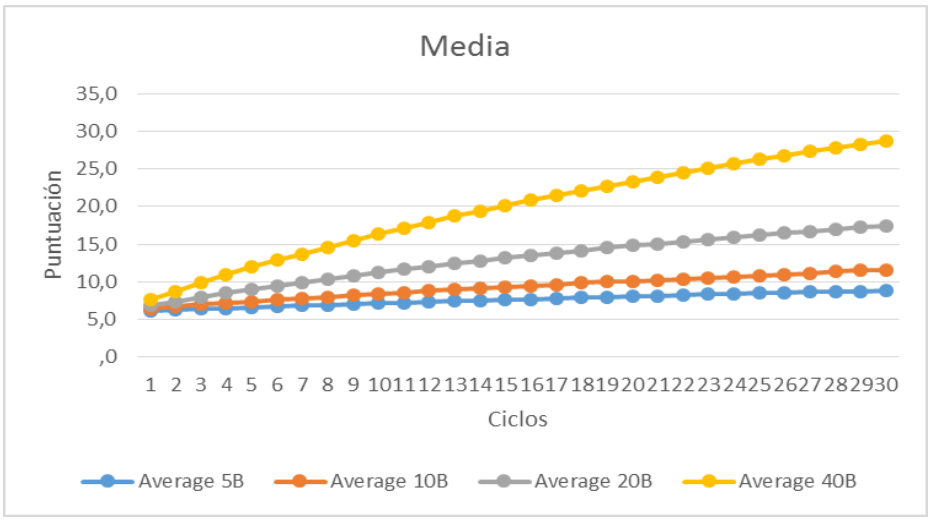

Figura 0-5: Media con 30 ciclos

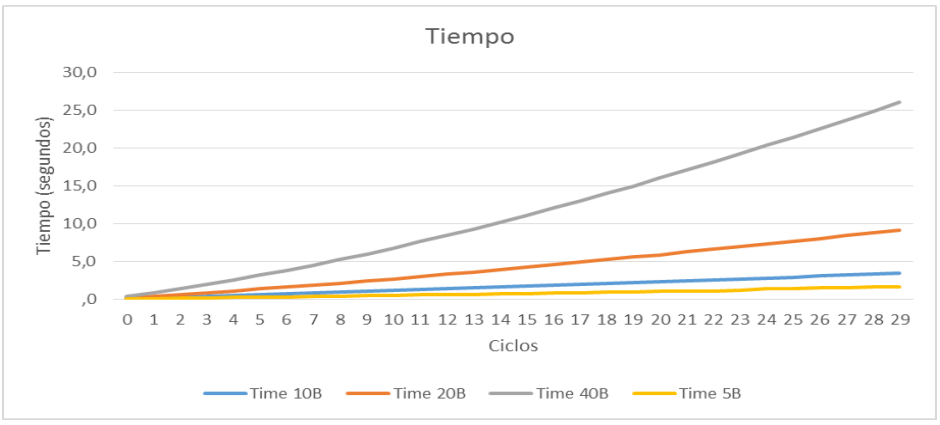

Figura 0-6: Tiempo con 30 ciclos

Número de ciclos: 50
Número de Abejas: 5, 10, 20, 40

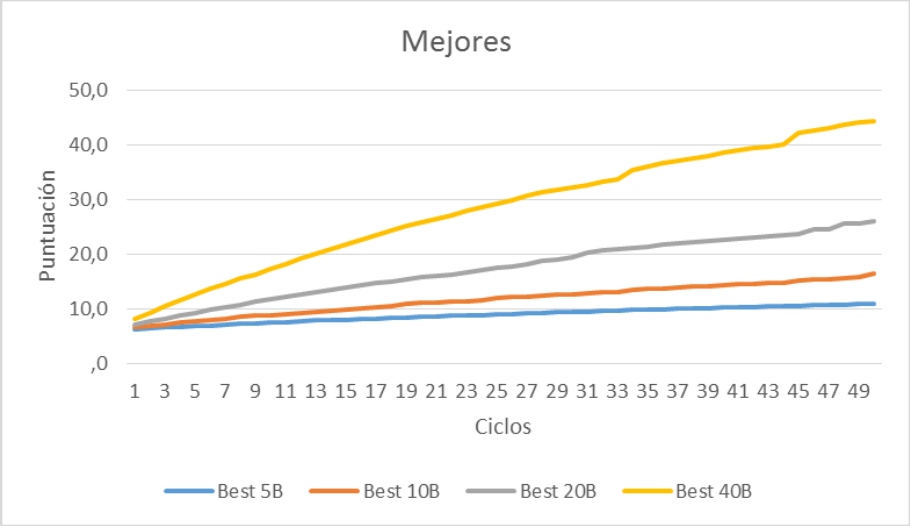

Figura 0-7: Mejores con 50 ciclos

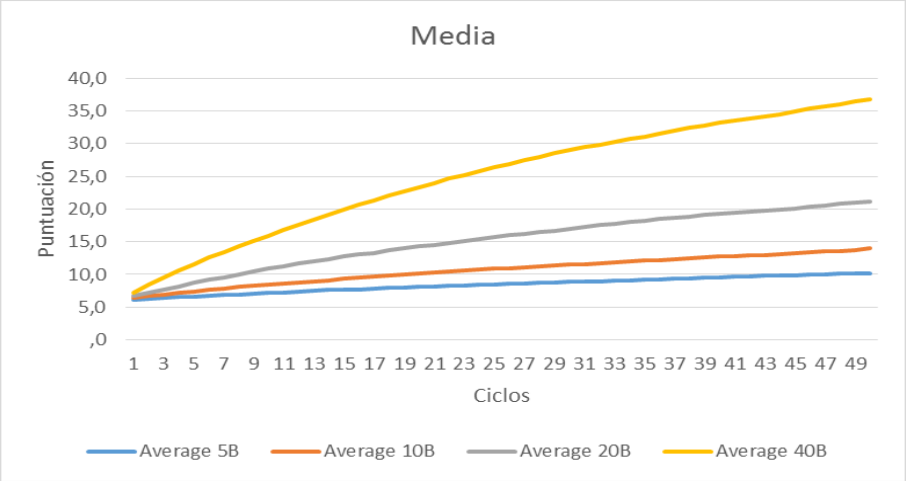

Figura 0-8: Media con 50 ciclos

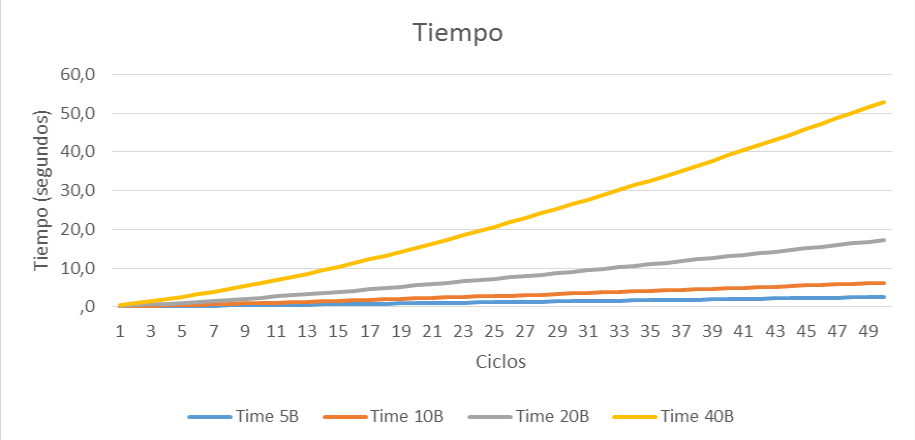

Figura 0-9: Tiempo con 50 ciclos

Número de ciclos: 100
Número de Abejas: 5, 10, 20, 40

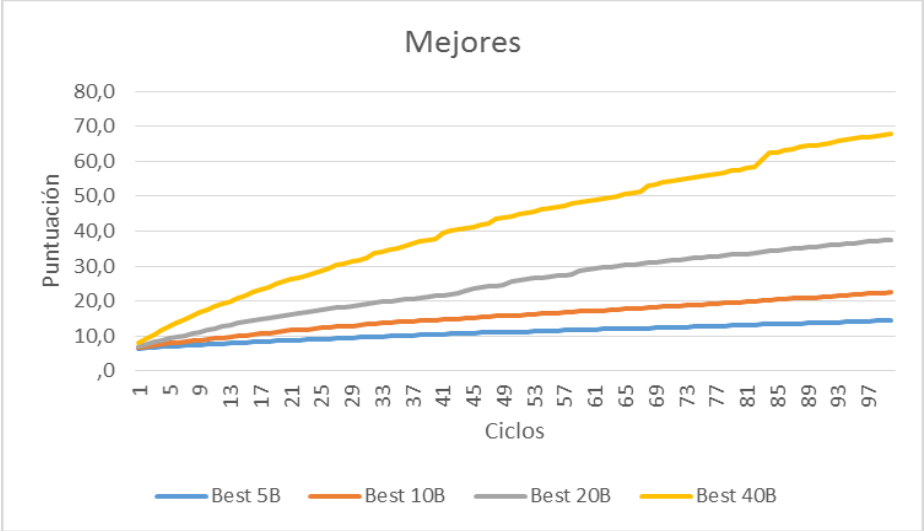

Figura 0-10: Mejores con 100 ciclos

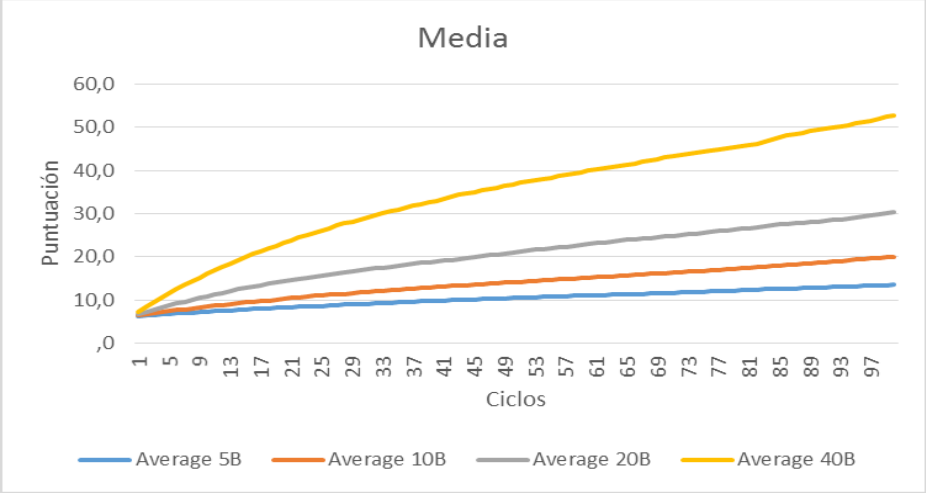

Figura 0-11: Media con 100 ciclos

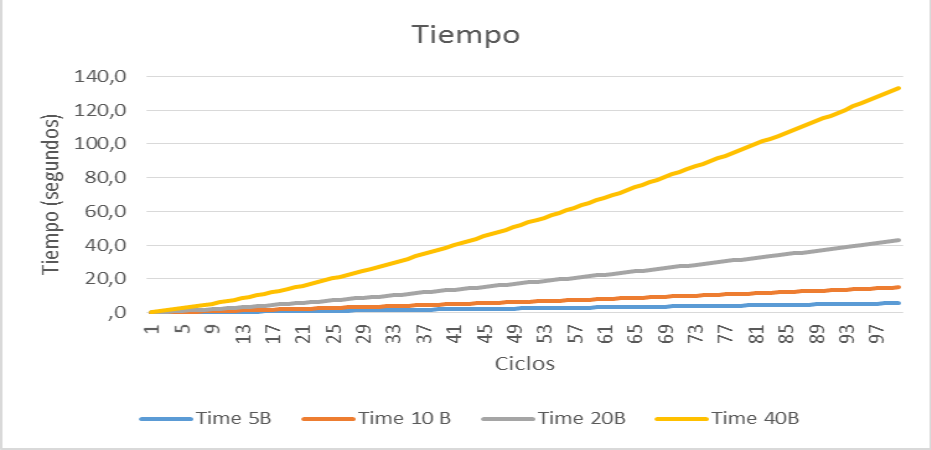

Figura 0-12: Tiempo con 100 ciclos

Número de ciclos: 200
Número de Abejas: 5, 10, 20, 40

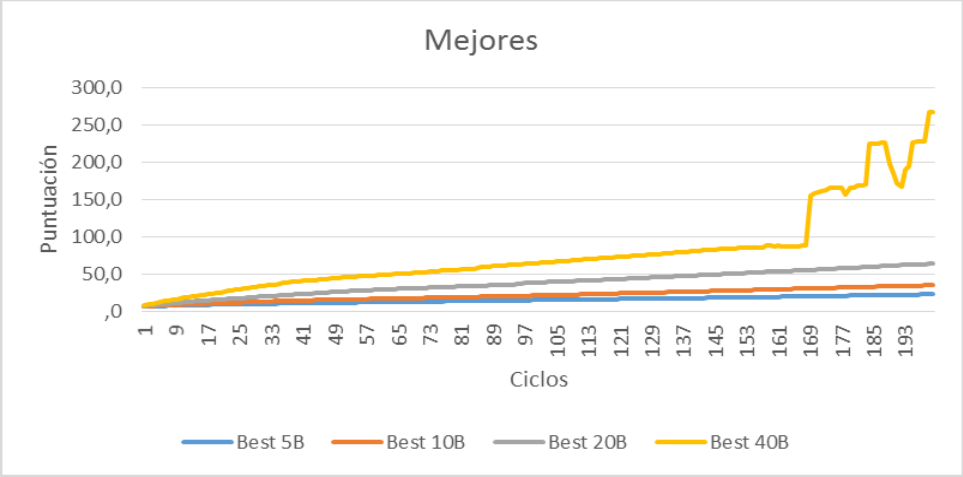

Figura 0-13: Mejores con 200 ciclos

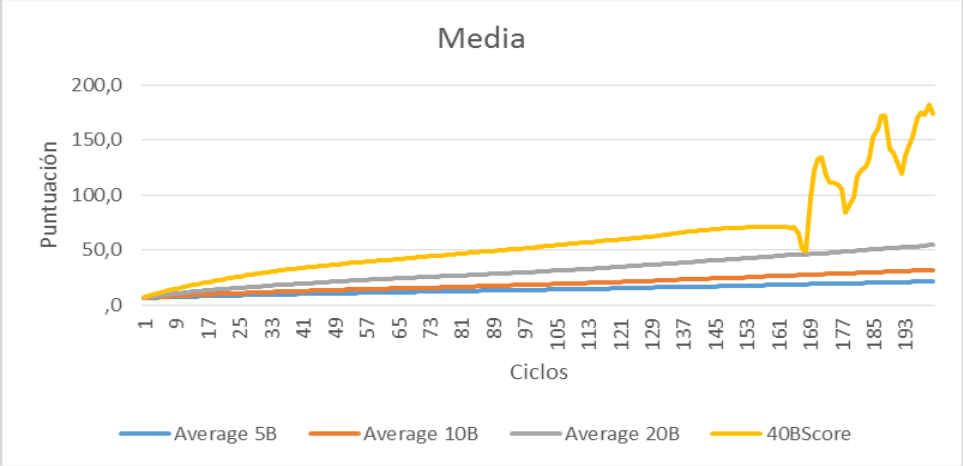

Figura 0-14: Media con 200 ciclos

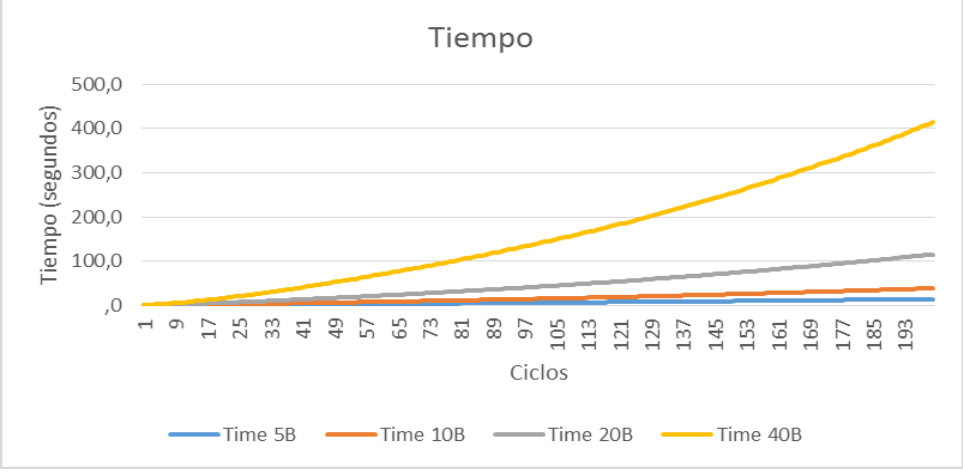

Figura 0-15: Tiempo con 200 ciclos

Número de ciclos: 500
Número de Abejas: 5, 10, 20, 40

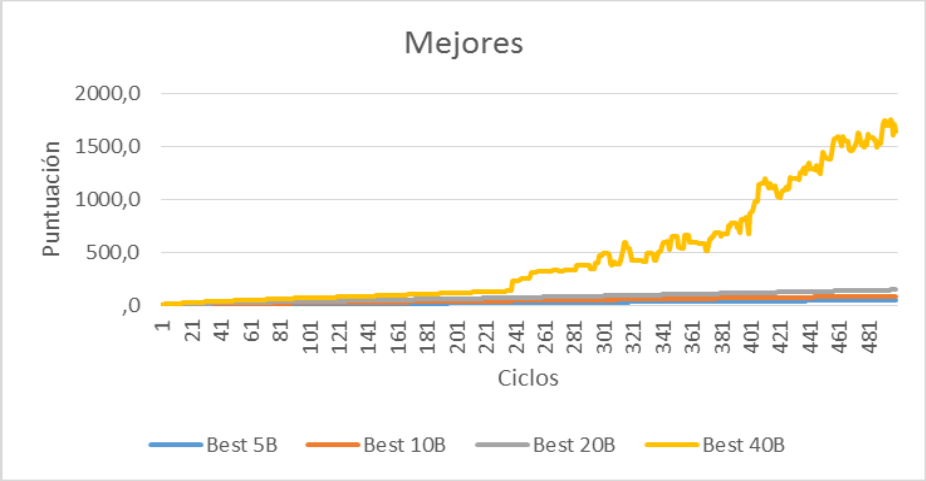

Figura 0-16: Mejores con 500 ciclos

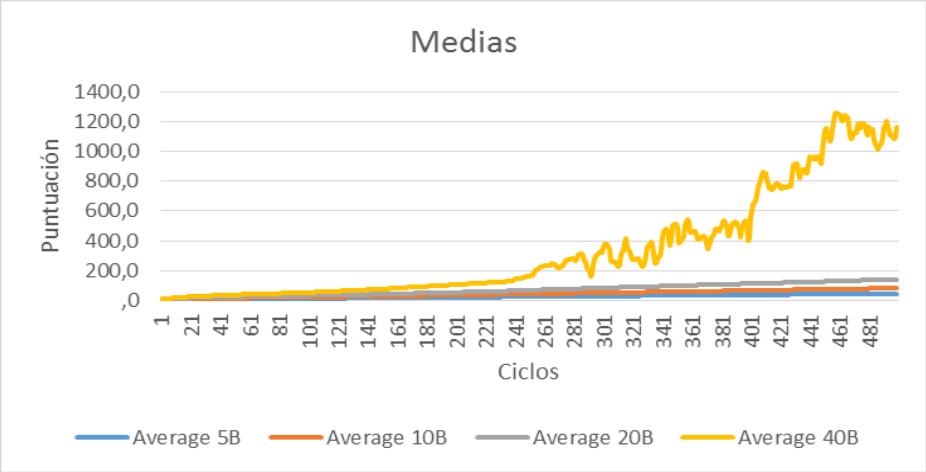

Figura 0-17: Media con 500 ciclos

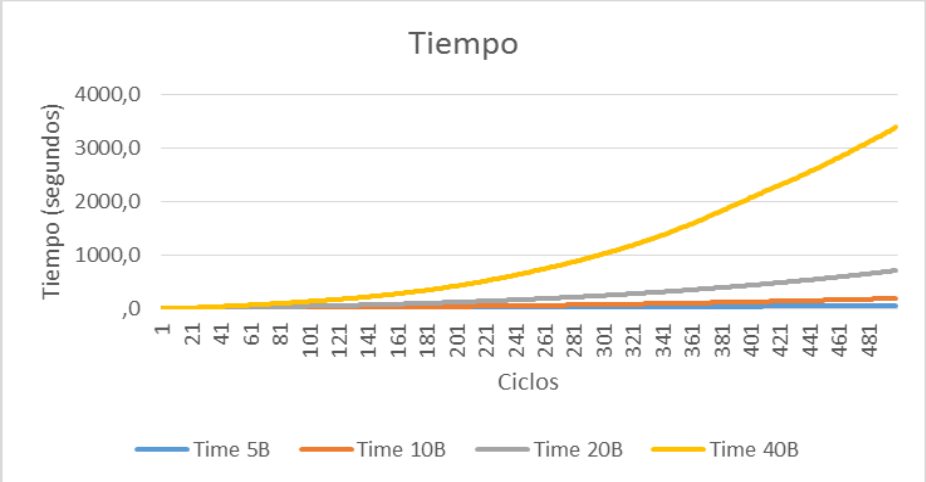

Figura 0-18: Tiempo con 500 ciclos

Número de ciclos: 1000
Número de Abejas: 5, 10, 20, 40

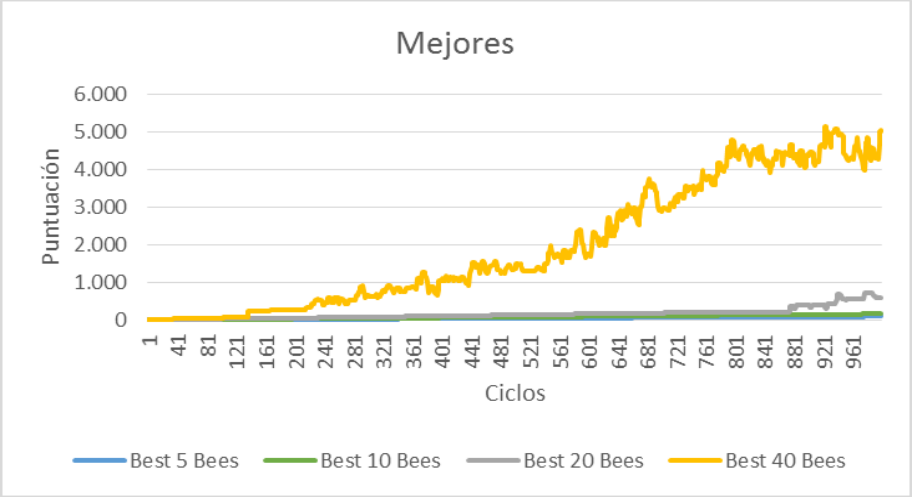

Figura 0-19: Mejores con 1000 ciclos

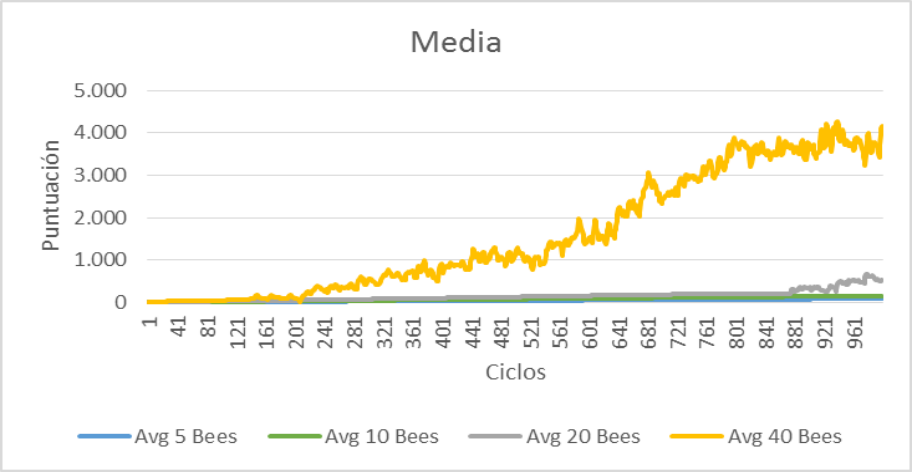

Figura 0-20: Media con 1000 ciclos

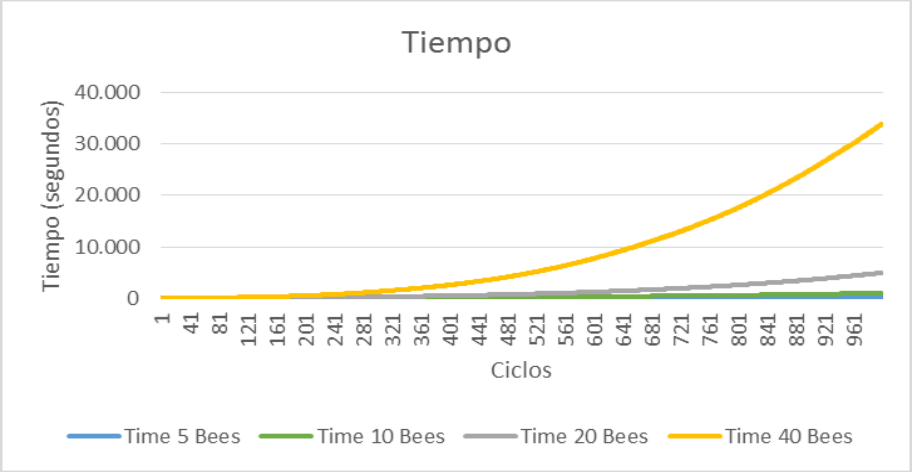

Figura 0-21: Tiempo con 1000 ciclos

Número de ciclos: 500
Número de Abejas: 20, 40, 80

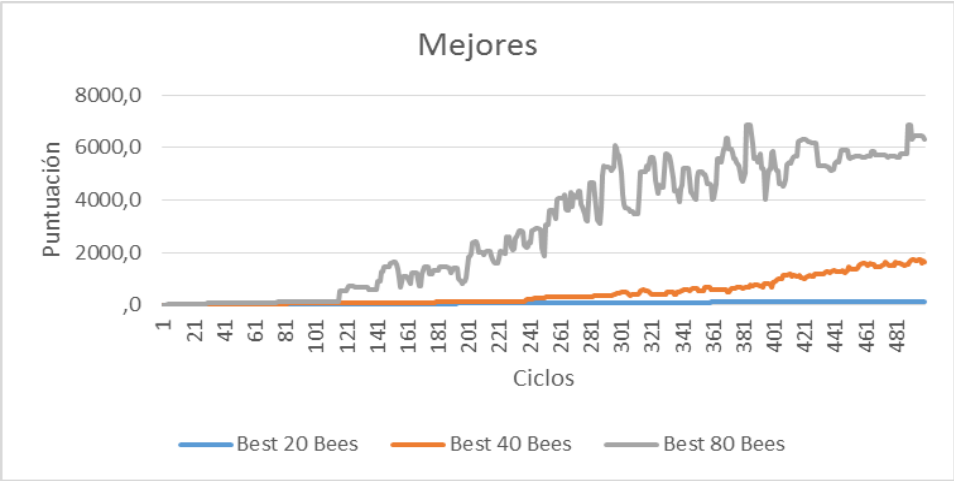

Figura 0-22: Mejores 500 ciclos (Prueba Abejas)

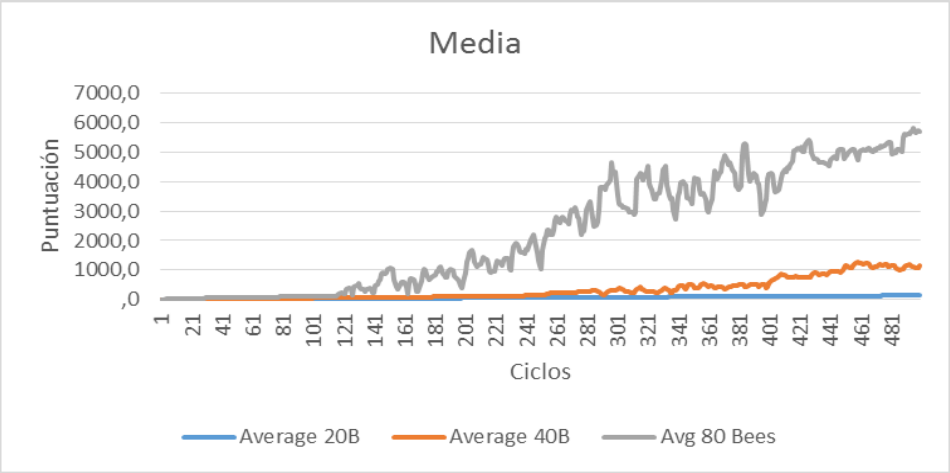

Figura 0-23: Media 500 ciclos (Prueba Abejas)

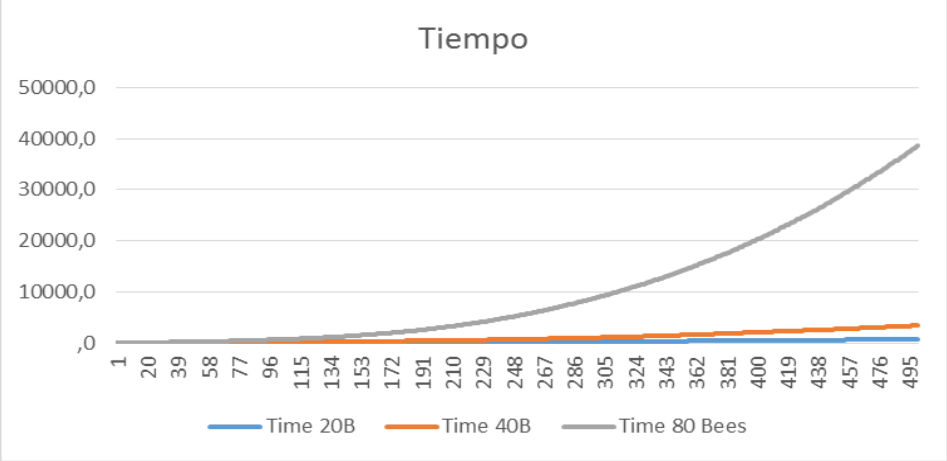

Figura 0-24: Tiempo con 500 ciclos (Prueba Abe